Markt&Technik WORKSHOP PC

Andreas Maslo

BASIC POWER

Erfolgreich starten – sicher nutzen.

Power Basic

Programm ausdrucken siehe Seite 166

File Write To -> PRN eingeben

WORKSHOP PC

Herausgegeben von .TXT Publishing GmbH, München

Andreas Maslo

POWER Basic

Erfolgreich starten – sicher nutzen

Markt&Technik Verlag AG

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Maslo, Andreas:
Power Basic : erfolgreich starten – sicher nutzen /
Andreas Maslo. –
Haar bei München : Markt-u.-Technik-Verl., 1991
(Workshop PC)
ISBN 3-87791-192-7

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Power Basic ist ein eingetragenes Warenzeichen von Kirschbaum Software GmbH.

12 11 10 9 8 7 6 5 4 3 2 1

94 93 92 91

ISBN 3-87791-192-7

© 1991 by Markt&Technik Verlag Aktiengesellschaft,
Hans-Pinsel-Straße 2, D-8013 Haar bei München/Germany
Alle Rechte vorbehalten
Einbandgestaltung: Grafikdesign Heinz Rauner
Herstellung: Werner Leidl
Lektorat: .TXT Publishing GmbH, München
Dieses Produkt wurde mit Desktop-Publishing-Programmen erstellt
und auf der Linotronic 300 belichtet
Druck: Paderborner Druck Centrum, Paderborn
Printed in Germany

Inhaltsverzeichnis

Vorwort 9

Kapitel 1: Einführung 11

1.1 Programmieren mit Power Basic 11
1.2 Wie Sie mit diesem Buch arbeiten 13
1.3 Schreibweisen 15
1.4 Hardware- und Software-Anforderungen 16
1.5 Installieren und Einrichten 17
1.6 Grundlagen zu Power Basic 22

Kapitel 2: Tutorium 33

2.1 Erste Sitzung: Das Konzept der Einnahmen-/Ausgabenverwaltung 33
2.2 Zweite Sitzung: Wahl einer Arbeitsdatei für Buchungssätze 53
2.3 Dritte Sitzung: Dateneingabe der Einnahmen und Ausgaben 66
2.4 Vierte Sitzung: Datenverwaltung mit Direktzugriffsdateien 79
2.5 Fünfte Sitzung: Datenauswertung 89
2.6 Sechste Sitzung: Druckerausgabe – Daten schwarz auf weiß 101
2.7 Siebte Sitzung: Daten grafisch dargestellt 113
2.8 Achte Sitzung: Optimierung durch Fehlerbehandlung 129
2.9 Neunte Sitzung: Ein Hilfesystem gefällig? 138
2.10 Zehnte Sitzung: Das Maschinenprogramm entsteht 151

Kapitel 3: Know-how 157

3.1	Fehlerbeseitigung	157
3.2	Erste Hilfe: Was tun bei drohendem Datenverlust?	164
3.3	Kniffe, Tips und andere Spezialitäten	165
3.4	Sinnvolle Hardware-Erweiterungen	170
3.5	Der Kommandozeilen-Compiler	171
3.6	Nützliche Zusatz-Software	172

Kapitel 4: Referenz 175

4.1	Menüreferenz	175
4.1.1	Das Menü »File«	176
4.1.2	Das Menü »Edit«	178
4.1.3	Das Menü »Run«	178
4.1.4	Das Menü »Compile«	179
4.1.5	Das Menü »Options«	180
4.1.6	Das Menü »Debug«	186
4.1.7	Das Menü »Break/Watch«	188
4.2	Sprachreferenz	189
4.2.1	Compiler-Anweisungen	190
4.2.2	Compiler-Daten und Variablendeklarationen	193
4.2.3	Dateien	195
4.2.4	Datenfeldoperationen	199
4.2.5	DOS-Anweisungen	201
4.2.6	Druckersteuerung	204
4.2.7	Eingabe- und Ausgabeanweisungen	205
4.2.8	Ereignis- und Hardware-Verfolgung	206
4.2.9	Fehlerbehandlung	210
4.2.10	Grafikanweisungen	211
4.2.11	Konvertierfunktionen	213
4.2.12	Mathematische Funktionen	219
4.2.13	Musik	222
4.2.14	Programmablaufsteuerung	222
4.2.15	Speicherverwaltung und Speicherfunktionen	229
4.2.16	Stringoperationen	232
4.2.17	Textbildschirm	235
4.3	Schnellinformation: Wie kann ich ...?	236

Anhang 245

A Quellen zum Programm HAUSHALT 245
B Glossar 260
C Tabellen 262
D Literaturhinweise 265

Stichwortverzeichnis 267

Vorwort

Workshop – die Buchreihe für den engagierten Computerbesitzer – wurde nach völlig neuem Konzept entwickelt, das sowohl die Möglichkeiten moderner Computerprogramme als auch aktuelle didaktische Methoden berücksichtigt. Die Aufgabenstellung:

Nicht einfach die Funktionsvielfalt eines Programmpakets zu demonstrieren, sondern vor allem die Fähigkeit der praktischen Umsetzung zu trainieren. Was herauskam, ist ein ausgefeiltes Praxiskonzept: mit fundiertem Startwissen, illustriertem Tutorium, zahlreichen Übungsideen, nützlichen Tips und Kniffen, Fehlerarten und Lösungsvorschlägen sowie knappem Referenzteil.

Das Herzstück eines Workshops ist das Tutorial. Es ist in zehn Sitzungen organisiert, und so angelegt, daß ein nützliches Programmprodukt mit allem Hintergrundwissen hergestellt wird. Innerhalb kürzester Zeit sind daher praktische, transferierbare Arbeitsergebnisse sichtbar. Dann – bei steigender Erfahrung – bleiben Know-how und Referenz die nützlichen Ratgeber für die tägliche Praxis. Workshop – die Buchreihe, die mit Ihrem Wissen und Erfahrungsschatz mitwächst.

Der Autor **Andreas Maslo** trägt mit seinen Erfahrungen als freiberuflicher Ingenieur und EDV-Autor wesentlich dazu bei, daß Sie *Power Basic* von Anfang effektiv nutzen und auch noch Spaß dabei haben. Bedanken möchte er sich bei Herrn Schlaak vom Verlag Markt & Technik und bei Frau Baumann von .TXT-Publishing, ohne die dieses Buch nicht realisierbar gewesen wäre. Besonderer Dank gilt seiner Frau für ihre Unterstützung und Hilfe bei diesem Projekt.

Wir hoffen, daß Ihnen dieses Buch gefällt und weiterhilft. Dennoch – es gibt nichts, was man nicht noch verbessern könnte. Teilen Sie uns doch bitte Ihre Meinung zu Buch und Programm mit. Sie helfen damit anderen Lesern und uns, das Workshop-Konzept noch besser nach Ihren Bedürfnissen auszurichten. Vielen Dank.

Die Herausgeber

.TXT Publishing GmbH

Kapitel 1: Einführung

1.1 Programmieren mit Power Basic

Basic wurde als Programmiersprache für Lehrzwecke entwickelt. War Basic (»**B**eginners' **A**ll Purpose **S**ymbolic **I**nstruction **C**ode«) zu Beginn lediglich als Interpreter-Dialekt erhältlich, so gibt es heute zunehmend mehr Compiler-Dialekte, die sich durch ihre strukturierte und modulare Programmierung auszeichnen. Die Leistungsfähigkeit kann durchaus mit anderen Programmiersprachen wie z.B. C oder Pascal konkurrieren. Im Gegensatz zu einem Interpreter wird ein Programm bei einem Compiler nicht zeilenweise, sondern komplett vor der Programmausführung zu einem eigenständig ablauffähigen Maschinenprogramm übersetzt. Dadurch sind Programme schneller in der Ausführung und ohne den eigentlichen Compiler lauffähig. Auch bei Power Basic handelt es sich um einen Compiler, der aus einer Weiterentwicklung des Turbo-Basic-Compilers entstanden ist. Es ist einer der preiswertesten und leistungsfähigsten Basic-Compiler auf dem derzeitigen Software-Markt.

Leistungsmerkmale von Power Basic

Wie alle anderen populären Programmierwerkzeuge, verfügt auch Power Basic über eine integrierte Entwicklungsumgebung, die sämtliche Dienstprogramme, wie Editor, Compiler, Linker, Debugger und Hilfesystem unter einer einheitlichen Benutzeroberfläche vereint. Innerhalb dieser Oberfläche werden Sie Ihre Programme schreiben, bearbeiten, testen und letztendlich in Maschinensprache übersetzen. Ein wichtiges Qualitäts-

merkmal von Software ist, wie eh und je, die mitgelieferte Dokumentation. Hier nimmt Power Basic eine Spitzenstellung ein. Es wird neben einem Benutzerhandbuch, das den Umgang mit der Entwicklungsumgebung, die Sprachsyntax und wesentliche Programmierthemen erläutert, ein Referenzhandbuch mitgeliefert, in dem sämtliche Power-Basic-Befehle in alphabetischer Reihenfolge ausführlich beschrieben werden. Einige Besonderheiten machen Power Basic auch für umfassende professionelle Programmentwicklungen zu einem geeigneten Werkzeug. Es bestehen vielfältige Möglichkeiten, um die Code-Generierung zu beeinflussen. So kann z.B. spezieller Maschinencode für einen Coprozessor oder den 80286-Hauptprozessor wahlfrei erzeugt werden und nicht benötigte Module aus der Runtime-Bibliothek brauchen in ein Maschinenprogramm nicht mit eingebunden zu werden. Binär kodierte Dezimalzahlen und Zeichenketten, die den gesamten Speicher, also mehr als 64 Kbyte belegen können, sind weitere, besonders herausragende Merkmale. Auch wenn Sie mit diesen Leistungsmerkmalen im Moment noch nicht viel anfangen können, so werden Sie diese im Verlaufe Ihrer Programmierprojekte sicherlich nicht mehr missen wollen.

Ein Basic für alle

Power Basic ist eine Programmiersprache, die sowohl für Anfänger als auch für professionelle Programmierer gleichermaßen geeignet ist. Kleine Hilfsprogramme lassen sich ebenso programmieren wie große komplexe Anwendungsprogramme. Der große Wortschatz mit umfangreichen DOS-Befehlen, Routinen zur Manipulation von Datenfeldern, vielfältigen Stringmanipulationen und Funktionen zur Speicherverwaltung machen die Programmierung zum Vergnügen. Mit Power Basic können Sie Ihre eigene Textverarbeitung, Dateiverwaltung oder Ihr eigenes Grafik- oder Kommunikationsprogramm erstellen. Programme, die ursprünglich für den GW-Basic-Interpreter, der in Verbindung mit dem MS-DOS-Betriebssystem ausgeliefert wird, entwickelt wurden, können ohne große Änderungen von Power Basic übersetzt werden.

Warum programmieren?

Obgleich für fast jeden Anwendungsbereich entsprechende Software erhältlich ist, ist auch heute die Anschaffung einer Programmiersprache wie Power Basic durchaus sinnvoll. Sie können sich Ihre dringend benötigte Software, z.B. kleine Dienstprogramme, ohne zusätzliche Kosten selbst und vor allen Dingen genau nach Ihren eigenen Bedürfnissen

entwickeln. Bedenken Sie, daß Sie bei Ihren eigenen Programmen immer über den Quelltext verfügen und so Anpassungen, die etwa durch erweiterte Hardware-Bedingungen notwendig werden, selbst vornehmen können. Daß die Programmierung eines optisch ansprechenden und umfangreichen Programms gar nicht so schwierig ist, soll dieses Buch demonstrieren.

1.2 Wie Sie mit diesem Buch arbeiten

Aufbau des Buches

Das Buch besteht im wesentlichen aus fünf Teilen. Diese sollen Sie sowohl mit dem Umgang der Benutzeroberfläche von Power Basic als auch mit der Anwendung der Programmiersprache vertraut machen.

Einführung

In dem ersten Kapitel erläutern wir zunächst den Einstieg in Power Basic. Dazu werden wir neben den Hardware- und Softwareanforderungen die Installation und die wichtigsten Elemente der Programmiersprache Power Basic besprechen.

Tutorium

Ist das Herzstück eines jeden Workshops. Um Power Basic möglichst anschaulich darzustellen, wird im Laufe des Tutoriums eine »Einnahmen-/Überschußverwaltung« mit grafischer Oberfläche programmiert. So entsteht eine vielfältige Sammlung von nützlichen Bausteinen, die Sie in jedem anderen Programm benutzen können. Das Tutorium, das aus insgesamt zehn Sitzungen besteht, demonstriert die wichtigsten Spracheigenschaften von Power Basic. Ferner wird eine systematische Anleitung zu einer strukturierten und modularen Programmierung gegeben. Jede Sitzung besteht aus einem Informationsteil, der die Grundlagen erarbeitet, einem Praxisteil mit Schritt-für-Schritt-Lösungen und einem Workshop, der zur Kontrolle des Erarbeiteten dient und Hinweise zum Üben gibt.

Mit den Erkenntnissen, die Sie in diesem Kapitel gewinnen, werden Sie später selbst in der Lage sein, leistungsfähige Programme zu entwickeln.

Know-how

Hier werden spezielle Themen zu Power Basic behandelt. Neben der Fehlerbeseitigung und besonderen Tips, stellen wir insbesondere den Kommandozeilen-Compiler und Toolboxen zu Power Basic vor.

Referenz

Dieses Kapitel besteht aus drei Teilen. In der Menüreferenz werden die Power-Basic-Menübefehle erklärt, und in der Sprachreferenz sind die Power-Basic-Anweisungen themenorientiert aufgeführt. Zusätzlich besprechen wir einige Standardaufgaben und ihre Lösungsmöglichkeiten.

Anhang

Im Anhang werden Begriffe erläutert, die in Verbindung mit der Programmierung wichtig sind. Ferner finden Sie dort die kompletten Quelltexte und einige wichtige Tabellen.

Befehlskarte

Das Allerwichtigste auf einen Blick.

Kleine Leseanleitung

Haben Sie Power Basic bereits installiert und sind mit den Grundlagen der Sprache vertraut, so können Sie direkt mit dem Tutorium beginnen. Dieses ist chronologisch aufgebaut, daher ist es nicht sinnvoll, zwischen den einzelnen Sitzungen zu springen. Arbeiten Sie das Kapitel Sitzung für Sitzung durch und versuchen Sie, die abgedruckten Programme nachzuvollziehen und durch die vorgeschlagenen Übungen zu erweitern. Wenn Sie das Programm nicht in der Gesamtheit nutzen wollen, können Ihnen einzelne Routinen dennoch Anregungen für eigene Programme geben. Daher wurde bei unserem Programmierprojekt »Haushalt« auf die strukturierte und modulare Programmierung besonderer Wert gelegt. Haben Sie diesen Arbeitsteil unseres Buches einmal durchgearbeitet, kann er auch als Nachschlagewerk für spätere Programmieraufgaben dienen. Mit den Kapiteln, die spezielle Informationen und Referenzen enthalten,

werden Sie später in erster Linie arbeiten. Sie sollten sich nach dem Tutorium unbedingt mit dem Kapitel Know-how befassen, da es einige sehr sinnvolle Hinweise für Ihr weiteres Programmieren enthält.

1.3 Schreibweisen

Da Power Basic in der derzeitigen Programmversion 2.10 nicht über eine Mausunterstützung verfügt, werden sämtliche Eingaben und Menübefehle durch die Tastatur eingegeben bzw. aktiviert. Um eine Menüfolge in unserem Buch hervorzuheben, wird diese in kursiver Schrift dargestellt. Auch Variablennamen, Basic- und Compiler-Anweisungen sowie Programmaufrufe werden durch Kursivschrift hervorgehoben. Dateinamen und Verzeichnisnamen werden in Großbuchstaben gedruckt.

Beispiele:

Menüfolgen:

File → Write To

Tastaturbefehle:

[Return]	Eingabetaste drücken
[Alt] + [F1]	[Alt]- und [F1]-Taste gleichzeitig drücken
[F1]	Funktionstaste [F1] drücken

Bei diversen Funktionen öffnen sich Eingabefenster, durch die bestimmte Angaben vom Programmierer abgefragt werden. Diese Eingaben werden erst bei Betätigung der [Return]-Taste übernommen, bei Betätigung der [Esc]-Taste werden sie ignoriert.

Folgende Symbole werden verwendet:

☞ Ein Hinweis ist eine Empfehlung oder ein Rat. Wenn Sie ihn ignorieren oder vergessen, passiert nicht allzuviel. Im schlimmsten Fall meldet der Compiler einen Fehler, den Sie korrigieren müssen.

⚠ **Wenn dieses Symbol auftaucht, ist Gefahr in Verzug. Es gibt zum Beispiel Tippfehler, die der Compiler nicht erkennt. Bei sollen Fehlern läuft das Programm falsch oder stürzt ab.**

1.4 Hardware- und Software-Anforderungen

Power Basic ist auf jedem IBM PC/XT/AT oder PS/2 sowie Kompatiblen mit 640 Kbyte Hauptspeicher und dem Betriebssystem MS-DOS, ab Version 2.0 oder dazu kompatiblen Betriebssystemen, wie z.B. DR-DOS, lauffähig. Beachten Sie, daß die Netzwerkunterstützung erst ab MS-DOS 3.1 verfügbar ist. Arbeiten Sie z.B. mit MS-DOS 2.11, können Sie keine Netzwerk-Befehle verwenden.

Sämtliche Hardware-Komponenten des Rechners werden von Power Basic nahezu optimal unterstützt. Die Programmentwicklung ist dennoch auch auf einem kleinen PC mit nur einem Diskettenlaufwerk möglich. In diesem Fall starten Sie Power Basic zunächst von einer Diskette und legen anschließend eine Arbeitsdiskette für Ihr Programm ein. Falls Sie allerdings nur über Diskettenlaufwerke mit einer Kapazität von 360 Kbyte verfügen, werden Sie sich zunächst genauer mit der Installation befassen müssen. Darauf werden wir aber später noch eingehen. Je besser Ihr Rechner ausgestattet ist, desto mehr haben Sie von Power Basic. Sie können zwar auch mit einem PC ohne Coprozessor Programme mit Coprozessor-Unterstützung bzw. ohne 80286-Hauptprozessor Code für den 80286-Hauptprozessor erzeugen, lauffähig sind diese Programme aber nicht mehr auf diesem PC. Auch läßt sich die optimale Grafikauflösung nur mit einer installierten VGA-Grafikkarte ausnutzen. Wenn Sie keine Grafikprogramme programmieren wollen, können Sie allerdings auf eine Grafikkarte verzichten.

Zusammenfassend läßt sich also sagen, daß Power Basic kaum Anforderungen an die Hardware stellt. Lediglich ausreichend Arbeitsspeicher und das richtige Betriebssystem sind notwendige Kriterien für den Einsatz. Für ein einfacheres Arbeiten sollte eine Festplatte und zum Ausnutzen sämtlicher Sprachanweisungen eine VGA-Grafikkarte vorhanden sein. Für die Beschleunigung von rechenintensiven Programmen kann aber auch die Anschaffung eines Coprozessors sinnvoll sein. Power Basic ist einer der flexibelsten Basic-Compiler, der derzeit erhältlich ist. Keiner unterstützt kleine Rechner so gut, wie gut ausgestattete.

Lieferumfang

Power Basic 2.10 wird in der deutschen Fassung mit einem Benutzerhandbuch und einem Referenzhandbuch ausgeliefert. Beide Bände sind jeweils ca. 400 Seiten stark und enthalten alle wesentlichen Informationen, die Anfänger und Fortgeschrittene für die Programmentwicklung benötigen. Die Software wird sowohl auf einer 5¼"-Diskette (1.2 Mbyte) als auch auf zwei 3½"-Disketten (720 Kbyte) ausgeliefert. Die Menüstruktur enthält ausschließlich englische Begriffe, lediglich die Hilfsinformationen sind in deutsche Sprache übersetzt. Nach einer kurzen Eingewöhnungszeit ergeben sich hierdurch allerdings keine Probleme. Schließlich sind auch alle Befehle der Basic-Syntax aus der englischen Sprache entnommen. Für die Installation benötigen Sie entweder nur die 5¼"-Diskette oder die 3¼"-Disketten. Erläuterungen zu den englischen Menüs sind in diesem Buch im Referenzteil enthalten.

1.5 Installieren und Einrichten

Bevor Sie Power Basic auf Ihrem Rechner installieren, sollten Sie zunächst Sicherungskopien des Programms herstellen. Sollte diese Arbeitskopie später einmal beschädigt werden, so haben Sie immer noch die Originaldisketten. Das Anfertigen von Sicherungskopien wird auch ausdrücklich von den Software-Herstellern empfohlen. Um eine Sicherungskopie zu erstellen, geben Sie folgendes nach dem Systemprompt C> ein:

```
DISKCOPY Lw: Lw: [Return]
```

Lw: steht dabei für das Diskettenlaufwerk, in das Sie wechselweise die Originaldiskette und die Diskette für die Arbeitskopie einlegen.

Installieren von Power Basic

Um Power Basic auf einem Rechner mit nur einem 360-Kbyte-Diskettenlaufwerk installieren zu können, sollten Sie Ihren Händler bitten, die Software zunächst auf das kleine PC-Format umzukopieren, da die mitgelieferten Originaldisketten nicht von diesem PC lesbar sind.

Im nachfolgenden werden wir die Installation auf einem Rechner mit nur einem Diskettenlaufwerk und alternativ auf einem Rechner mit einer Festplatte und einem Diskettenlaufwerk erläutern.

Installation auf Diskette

Eine Installation auf einer Diskette gestaltet sich je nach Diskettenformat (360 Kbyte oder 1.2 Mbyte) unterschiedlich. Nehmen wir an, Sie installieren das Programm auf einer 5¼"-Diskette mit 1.2 Mbyte. Fertigen Sie sich zunächst eine neue Sicherungskopie der Originaldiskette an. Anschließend löschen Sie die Beispieldateien, den Kommandozeilen-Compiler, den Sie nur für größere Programme benötigen, das Konfigurationsprogramm (natürlich erst nach der Installation) und die Datei TWORD.DAT. Die Befehle, die Sie nach dem Prompt eintasten, lauten dann wie folgt:

```
DISKCOPY A: A: [Return]
```

Nach dem Kopiervorgang sollten Sie darauf achten, daß die Arbeits- und nicht die Originaldiskette im Laufwerk liegt.

```
DEL *.BAS [Return]
DEL *.DAT [Return]
```

Anschließend können Sie Power Basic mit dem Programm PBINST nach Ihren eigenen Bedürfnissen konfigurieren. Das Programm PBINST können Sie dann ebenfalls löschen.

```
DEL PBINST.EXE [Return]
```

Auf der Diskette befinden sich nunmehr nur noch die Dateien PB.EXE, REGNAMES.INC und PBHELP.PBH. Verfügen Sie lediglich über das 360-Kbyte-Diskettenformat, dann sollten Sie sich eine Diskette erstellen, auf der nur das Hauptprogramm PB.EXE enthalten ist. In diesem Fall müssen Sie allerdings auf die integrierte Hilfsfunktion von Power Basic verzichten.

Installation auf einer Festplatte

Um das Programm auf einer Festplatte zu installieren, sollten Sie zunächst ein Verzeichnis für Power Basic einrichten. Anschließend wechseln Sie in dieses Verzeichnis und kopieren dorthin sämtliche Dateien der Originaldisketten. Dazu legen Sie Ihre Arbeitskopie bzw. Arbeitskopien in das Diskettenlaufwerk A: oder B: ein. Nehmen wir an, unser Verzeichnis soll den Namen »PB« erhalten und unser Festplattenlaufwerk hat den Laufwerksbuchstaben »C«.

Die Originaldiskette haben wir in Laufwerk A: eingelegt. Dann wären folgende Befehle auf Kommandozeilenebene hinter dem Systemprompt C> einzutippen:

```
MD PB      [Return]
CD PB      [Return]
XCOPY A:   [Return]
```

Nun können Sie Power Basic durch Eingabe von

```
PB [Return]
```

direkt aufrufen. Um Power Basic auch aus anderen Verzeichnissen starten zu können, sollten Sie über den *PATH*-Befehl in der AUTOEXEC.BAT den Verzeichnispfad global setzen. Hinweise hierzu entnehmen Sie bitte Ihren Betriebssystem-Handbüchern.

Konfigurieren von Power Basic

Mit Power Basic wird das Dienstprogramm PBINST.EXE mitgeliefert, mit dem Sie Power Basic an Ihre eigenen Bedürfnisse anpassen können. Notwendig ist die Konfiguration allerdings nicht.

```
Willkommen zum Installationsprogramm von PowerBASIC 2.1!

Dieses Programm erlaubt es Ihnen die Voreinstellungen für Compiler,
Umgebung und das Inhaltsverzeichnis von dem aus PowerBASIC jedesmal
starten wird zu treffen. Sie können ebenfalls die Farben für Menüs
und andere Informationen auf dem Schirm und den Videomonitor den Sie
benutzen einstellen. Sie können die Kommandos für den Editor ändern
und an Ihre Wünsche anpassen. Außerdem können Sie die Standardein-
stellungen für die Optionen im Editor wählen. Schließlich können Sie
in dem Programm noch die Größe für das Fenster Watch und Output in
der Integrierten Entwicklungsumgebung wählen.

Falls Sie dieses Programm benutzen, müssen Sie eine Kopie des PowerBASIC-
Compilers (PB.EXE) den Sie installieren wollen, im gleichen Inhalts-
verzeichnis wie das PBINST-Programm haben.

Sie können /B als Parameter für PBINST angeben um das Programm im Schwarz/
Weiß Modus zu starten. /C startet im Farbmodus. Wenn Sie keinen Parameter
angeben, wird der Videomodus automatisch festgestellt.

PowerBASIC Version 2.1 ist Copyright (c) 1990 by Robert S. Zale. Alle
Rechte vorbehalten. Distribution im deutschsprachigen Raum:
Kirschbaum Software GmbH, Kronau 15, W-8091 Emmering   Tel.: 08067/1016

  Drücken Sie ESC für Abbruch oder eine andere Taste zum Weitermachen...
```

Bild 1.5.1: Der Eröffnungsbildschirm von PBINST

Mit PBINST können Sie sowohl Farbeinstellungen als auch Tastaturbelegungen für den Editor verändern. Suchpfade lassen sich z.B. für Quelldateien und Units festlegen. Letztere, wie auch diverse Menüeinstellungen, sind ebenso über die Power-Basic-Oberfläche einstellbar.

Die durch PBINST gemachten Änderungen werden nicht in eine spezielle Konfigurationsdatei, sondern direkt in das Programm PB.EXE geschrieben. Sie sollten zunächst mit Power Basic gearbeitet haben, um Ihre optimalen Einstellungen zu kennen. Natürlich können Sie PBINST ohne große Gefahr aufrufen und sich anschauen, welche Einstellungen möglich sind, da vor dem Speichern der Informationen in das Hauptprogramm PB.EXE eine Sicherheitsabfrage erscheint.

Sie sollten PBINST erst anwenden, wenn Sie mit der Bedienung des Editors oder der Farbgebung nicht zufrieden sind, da die wichtigen Einstellungen, wie z.B. Menüvorgaben und Festlegung der Optionen, einfacher über das Hauptprogramm von Power Basic festzulegen sind. Diese können zudem in einer Konfigurationsdatei gespeichert werden, so daß die Informationen bei jedem Programmstart wieder aktivierbar sind. Ferner haben Sie auf diesem Wege die Möglichkeit, mehrere unterschiedliche Konfigurationsdateien anzulegen und die jeweils benötigte bei Bedarf zu laden.

```
                    Wählen Sie einen Installationspunkt

Wählen Sie einen Punkt und drücken Sie die entsprechende Taste:

1 - PowerBASIC's Compile Menü installieren
2 - PowerBASIC's Options Menü installieren
3 - PowerBASIC's Debug Menü installieren
4 - PowerBASIC's Editorkommandos einstellen
5 - PowerBASIC's Editorgrundeinstellungen wählen
6 - Videomodus wählen, der beim Start von PowerBASIC aktiv ist
7 - Farben einstellen, die PowerBASIC für die Menüs benutzt
8 - Einstellen der max. Größe von PowerBASIC's Watch/Output Fenster
9 - Installation beenden und Änderungen speichern (oder verwerfen)

                Drücken Sie 1, 2, 3, 4, 5, 6, 7, 8, oder 9...
```

Bild 1.5.2: Hauptauswahlmenü von PBINST

Der Vollständigkeit halber werden wir das Konfigurationsprogramm an dieser Stelle kurz erläutern. Nachdem Sie es mit

PBINST [Return]

gestartet haben, erscheint der Eröffnungsbildschirm, wie in Bild 1.5.1 dargestellt.

Beachten Sie, daß das Programm automatisch mit einer Fehlermeldung abgebrochen wird, wenn sich das Programm PB.EXE nicht im aktuellen Verzeichnis befindet. Durch Betätigung der [Return]-Taste gelangen Sie

zum Hauptmenü. Hier können Sie die Punkte auswählen, die Sie standardmäßig ändern wollen (Bild 1.5.2).

☞ Beachten Sie, daß Sie nie mit der Originalversion des Programms PB.EXE arbeiten, sondern mit der bereits installierten.

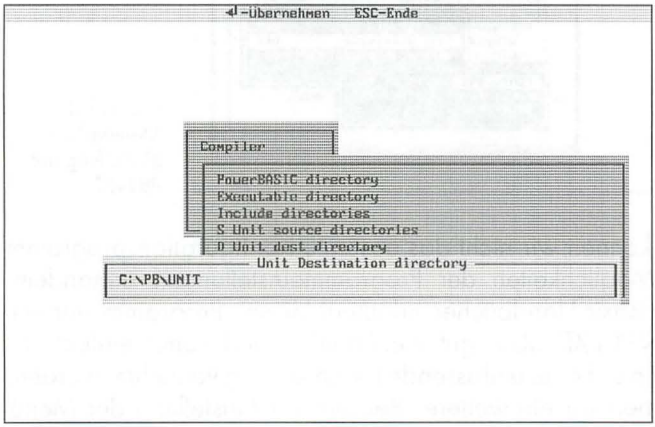

Bild 1.5.3: Suchpfad-Festlegung mit PBINST

Wenn Sie sehr viel programmieren, werden sich auf Dauer eine Menge von Quelldateien, Units usw. ansammeln. Arbeiten Sie auf einer Festplatte und wollen diese Dateien zunächst nicht löschen, so ist es sinnvoll, spezielle Verzeichnisse für diese Dateien anzulegen. Für Quelldateien könnten Sie z.B. ein Verzeichnis QUELLEN, für Units das Verzeichnis UNIT auf Betriebssystem-Ebene mit dem DOS-Kommando MD erstellen. Damit der Compiler und der Linker im Anschluß daran die Dateien in diesen Verzeichnissen finden kann, müssen Sie aus dem Hauptmenü den zweiten Menüpunkt *Power Basic's Option Menü installieren* anwählen, um die Optionen für den Compiler und Linker einzustellen.

Aus dem sich anschließend öffnenden Fenster wählen Sie mit der Cursorsteuerung (Pfeiltasten) den Menüeintrag *Directories* aus und quittieren diese Anwahl mit der [Return]-Taste. In einem weiteren Auswahlfenster erhalten Sie dann die Liste der einstellbaren Suchpfade. Nach einer beliebigen Anwahl erscheint ein Eingabefenster, in das Sie den gewünschten Suchpfad eingeben können. Der Bildschirm kann dann in etwa, wie in Bild 1.5.3 dargestellt, aussehen.

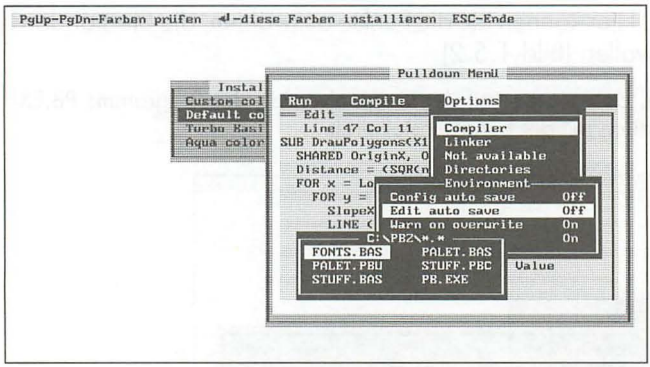

Bild 1.5.4: Menüfarbeneinstellung mit PBINST

An dieser Stelle können wir nicht das gesamte Konfigurationsprogramm mit sämtlichen Möglichkeiten der Programmeinstellungen behandeln. Auch die Power-Basic-Handbücher erläutern dieses Programm nur am Rande. Da PBINST.EXE aber gut menügeführt und somit einfach zu verstehen ist, kann auf eine umfassende Beschreibung verzichtet werden. In Bild 1.5.4 haben wir ein weiteres Beispiel zur Einstellung der Menüfarben abgebildet. Änderungen der Farben werden direkt in einem Power-Basic-Menü auf dem Bildschirm angezeigt.

1.6 Grundlagen zu Power Basic

An dieser Stelle wollen wir die wesentlichen Grundlagen besprechen, die zur Bedienung der Entwicklungsumgebung und zum Verständnis der Programmiersprache Power Basic erforderlich sind. Um Power Basic zu starten, tippen Sie auf der Kommandozeilenebene

PB [Return]

Im Anschluß daran erscheint der Eröffnungsbildschirm.

Die Benutzeroberfläche

Um mit der Oberfläche von Power Basic arbeiten zu können, brauchen Sie kein Handbuch, da Ihnen jederzeit eine programminterne Hilfsfunktion zur Verfügung steht, die sich mit der Funktionstaste [F1] aktivieren läßt. Befinden Sie sich in den Pull-down-Menüs und ein bestimmter Menüeintrag ist durch einen inversen Balken hervorgehoben, so erhalten Sie beim

Drücken der Taste [F1] genau zu diesem Menüeintrag eine Beschreibung. Wenn Sie sich im Editor befinden, stehen Ihnen allgemeine Informationen zur Verfügung. Um den Hilfe-Index angezeigt zu bekommen, müssen Sie die [F1]-Taste zweimal innerhalb des Editors betätigen. Aus diesem Index wählen Sie einen bestimmten Begriff, zu dem Sie nähere Erläuterungen benötigen (Bild 1.6.1).

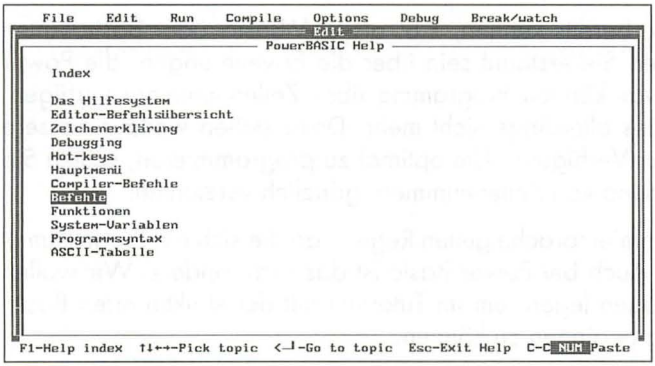

Bild 1.6.1: Hilfe-Index zur allgemeinen Hilfsfunktion von Power Basic

Auf diesem Wege können Sie sich auch Informationen zu beliebigen Befehlen von Power Basic auf dem Bildschirm anzeigen lassen, so daß Sie nicht bei jeder Kleinigkeit im Referenzhandbuch nachschlagen müssen. Die Hilfemeldung zur *SUB*-Anweisung ist in Bild 1.6.2 dargestellt.

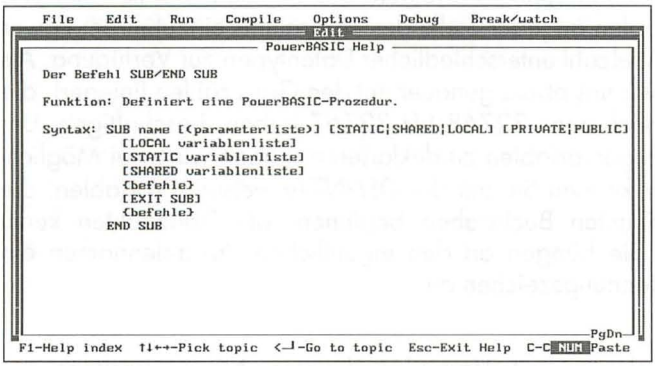

Bild 1.6.2: Hilfsinformation zur SUB/END-SUB-Anweisung

Für die Bedienung brauchen Sie ansonsten nur zu wissen, daß die jeweils aktiven Sondertasten in der untersten Bildschirmzeile angezeigt werden, daß Sie mit der Funktionstaste [F10] in die Menüsteuerung in der obersten

Bildschirmzeile gelangen und das eine Anwahl der einzelnen Menüeinträge durch die Cursorsteuerung (Pfeiltasten) erfolgt. Quittiert wird eine Menüanwahl durch [Return], ein Abbruch einer Eingabe bzw. einer Menüauswahl erfolgt in der Regel durch die [Esc]-Taste.

Die Sprachsyntax von Power Basic

Sollten Sie Basic bereits kennen, z.B. als GW-Basic- oder BasicA-Interpreter, so werden Sie erstaunt sein über die Erweiterungen, die Power Basic bietet. Zwar können Programme über Zeilennummern verfügen, notwendig ist dies allerdings nicht mehr. Dazu stehen weitaus bessere Anweisungen zur Verfügung. Um optimal zu programmieren, sollten Sie auf die Verwendung von Zeilennummern gänzlich verzichten.

Für jede Programmiersprache gelten Regeln, an die sich ein Programmierer halten muß. Auch bei Power Basic ist das nicht anders. Wir wollen nun die Grundlagen legen, um im Tutorium mit der strukturierten Basic-Programmierung beginnen zu können.

Datentypen

Variablen müssen je nach dem zugewiesenen Wert, den sie aufnehmen sollen, mit einem bestimmten Typ versehen werden. In anderen Programmiersprachen, z.B. C, Pascal oder Fortran, hat diese Deklaration immer am Programmbeginn zu erfolgen. Basic ist sehr viel flexibler und erlaubt es, Variablen an jeder Stelle des Programms einzuführen. Power Basic stellt eine Vielzahl unterschiedlicher Datentypen zur Verfügung. Als Beispiel wollen wir uns etwas genauer mit den Ganzzahlen (Integer), die einen Wertebereich von –32768 bis 32767 haben, beschäftigen. Um Variablen als Integervariablen zu deklarieren, haben wir zwei Möglichkeiten. Entweder können Sie mit der *DEFINT*-Anweisung Variablen, die mit einem bestimmten Buchstaben beginnen, als Ganzzahlen kennzeichnen, oder Sie hängen an den eigentlichen Variablennamen ein sogenanntes Erkennungszeichen an.

```
DEFINT A, B, C
```

In diesem Beispiel wären alle Variablen, die einen Namen besitzen, der mit einem a, b oder c beginnt, automatisch Ganzzahlen. Besser ist die zweite Methode. Dann erkennen Sie den Typ der Variablen an jeder Stelle des Quelltextes, an der diese eingesetzt wird. Nehmen wir an, die Variable *Alpha* soll als Ganzzahl deklariert werden. In diesem Fall

schreiben wir *Alpha%* in unser gesamtes Programm. Die nachfolgende Tabelle listet die unter Power Basic verfügbaren Datentypen, das jeweils dazugehörige Erkennungszeichen am Beispiel der Variablen *a* und den Wertebereich des Datentyps auf.

⚠ **Beachten Sie, daß gleiche Variablennamen mit unterschiedlichem Typkennzeichen vom Compiler als unterschiedliche Variablen behandelt werden. Beispiel: *Alpha%* und *Alpha!* sind unterschiedliche Variablen.**

Datentyp	Beispiel	Wertebereich
Integer	a%	–32.768 bis 32.767
Long-Integer	a&	–2.147.483.648 bis 2.147.4836.47
Quad-Integer	a&&	9.22×10^{18}
einfachgenau	a!	8.43×10^{-37} bis 3.37×10^{38}
doppeltgenau	a#	4.19×10^{-307} bis 1.67×10^{308}
erweitert	a##	3.4×10^{-4932} bis 1.2×10^{4932}
BCD-Festzahl	a@	9.99×10^{-63} bis 9.99×10^{63}
BCD-Fließzahl	a@@	9.99×10^{-63} bis 9.99×10^{63}
Strings	a$	bis zu 32750 Zeichen
Flex-String	a$$	flexible Zeichenkette

Tabelle 1.6.1: Power-Basic-Datentypen

Um die Variablendeklaration ein wenig verständlicher zu machen, wollen wir an dieser Stelle ein kleines Programm entwickeln, das zwei Ganzzahlen addiert. Die Zahlen sollen über die Tastatur eingelesen werden. Für die erste Zahl verwenden wir den Variablennamen *ErsterWert%* und für den zweiten Wert die Variable *ZweiterWert%*. Beachten Sie die hierbei gewählten Erkennungszeichen an den Variablennamen. Nun brauchen wir aber noch eine Variable, der das Ergebnis der Addition zugewiesen wird. Da die Summierung von 32000 und 32000 den Wertebereich der Ganzzahlen (Integer) überschreiten würde, müssen wir für die Ergebnisvariable einen Long-Integer-Datentyp wählen (Tabelle 1.6.1). Daher wird unser Ergebnis der Variablen *Summe&* zugewiesen. Demnach kann unser Programm etwa wie folgt aussehen:

```
INPUT "1.Wert: "; ErsterWert%
INPUT "2.Wert: "; ZweiterWert%
Summe& = ErsterWert% + ZweiterWert%
PRINT "Ergebnis der Addition: "; Summe&
```

Geben Sie in Programmen den Variablen aussagekräftige Namen, damit das Programm einfacher zu lesen ist. Ein Variablenname darf aus bis zu

255 Zeichen bestehen. Diese Länge sollten Sie allerdings nicht ausnutzen, da solche Variablennamen nur schlecht lesbar sind.

⚠ **Beachten Sie, daß vom Compiler Variablen ohne bestimmte Typzuweisung automatisch als einfachgenaue Fließkommavariablen eingestuft werden.**

Neben numerischen Variablen spielen in der Programmierung die Zeichenketten eine wesentliche Rolle (Tabelle 1.6.1). Sämtliche Texte werden als Zeichenketten verwaltet. Spezielle Anwendungen werden wir in unserem Tutorium behandeln.

Natürlich kennt auch Power Basic **Konstanten**. Der Wert einer Konstanten bleibt während des gesamten Programmlaufs unverändert. Die Zuweisung eines Wertes an eine Konstante darf innerhalb eines Programms nur einmal erfolgen. Einige Beispiele für Konstanten haben wir nachfolgend aufgelistet.

```
Stringkonstante                : "Test"
Integerkonstante dezimal       : 1%
                 hexadezimal   : &H1
                 binär         : &B00000001
Bezeichnete Integerkonstante   : %Werte
```

In Power Basic können auch sogenannte **Arrays** bzw. Datenfelder definiert werden. Haben wir zuvor einzelne Werte Variablen zugewiesen, können wir mit Hilfe der Felder komplette Listen verwalten. Datenfelder können für jeden Datentyp mit dem *DIM*-Befehl dimensioniert werden. Um zum Beispiel ein Datenfeld mit 20 Elementen zu erstellen, schreiben wir folgende Anweisung in unser Programm:

```
DIM Liste%(20) 'Ganzzahldatenfeld
```

Auf jedes Element des Datenfeldes kann mit einer Indexnummer zugegriffen werden. Das erste Element entspricht dann der Variablen *Liste%(1)*, das zweite Element *Liste%(2)* usw.

Ausdrücke und Operatoren

Variablen, Konstanten, Zeichenketten oder Datenfeldelemente können über Ausdrücke und Operatoren arithmetisch, relational oder logisch miteinander verknüpft werden. Am bekanntesten dürften hierbei die arithmetischen Operatoren sein, mit denen einfache mathematische Anweisungen durchgeführt werden. Den arithmetischen Operator für die Addition haben wir bereits in unserem Beispielprogramm kennengelernt.

```
arithmethische Operatoren: ^, -, *, /, \, MOD, +, -
relationale Operatoren   : <, <=, <>, >=, > , =
logische Operatoren      : AND, OR, XOR, EQV, IMP
```

Mit den Relationen können Sie Werte miteinander vergleichen. Das Ergebnis ist immer ein boolescher Wert, nämlich –1, wenn die Beziehung wahr (*TRUE*) ist, und 0, wenn die Beziehung falsch (*FALSE*) ist. Über die logischen Operatoren können Werte miteinander verknüpft werden. Eine Vielzahl der hier vorgestellten Operatoren werden wir in unserem Tutorium bereits in der Anwendung erleben.

Kontrollanweisungen

Um strukturiert programmieren zu können, sind spezielle Kontrollstrukturen erforderlich, mit denen die Programmablaufsteuerung beeinflußt werden kann. Die Kenntnisse dieser Strukturen ermöglichen einen einfacheren Einstieg in das Tutorium. Daher wollen wir an einfachen Beispielen diese Sprachelemente von Power Basic vorstellen.

Die *GOTO*-Anweisung ist der einfachste Befehl, um die Programmablaufsteuerung zu beeinflussen. Damit können Sie an eine beliebige Sprungmarke verzweigen. Bei häufigen *GOTO*-Anweisungen kann der Quelltext aber nur noch sehr schlecht nachvollzogen werden, weswegen Sie diesen Befehl, nur wenn unbedingt nötig, einsetzen sollten.

Beispiel:

```
GOTO weiter
PRINT "Hallo"
weiter:
END
```

In unserem Beispiel wird direkt am Programmanfang zu einer Programmendemarke verzweigt. Die Zeile zur Ausgabe des Textes *"Hallo"* auf dem Bildschirm wird also nie ausgeführt.

Mit Hilfe von **Schleifen** lassen sich bestimmte Arbeitsgänge mehrmals, zum Teil abhängig von einer bestimmten Bedingung, abarbeiten. Neben der *FOR-NEXT*-Schleife stellt Power Basic die *WHILE-WEND*- und *DO-LOOP*-Schleife zur Verfügung.

Allgemeiner Aufbau:

```
FOR Zaehler = Anfang TO Ende [STEP Schrittweite]
  ...
NEXT Zaehler
```

In unserer allgemeinen Formulierung ist *Zaehler* eine Schleifenvariable, in der Regel mit dem Datentyp Integer, *Anfang* ist der Startwert des Schleifenzählers, *Ende* der Endwert des Schleifenzählers und *Schrittweite* der Wert, um den die Zählvariable nach jedem Schleifendurchlauf erhöht wird. Wird *STEP Schrittweite* weggelassen, wird die Zählvariable jeweils um 1 erhöht.

Beispiele:

```
FOR x%=1 TO 100
  PRINT "Hallo"
NEXT x%

FOR x%=100 TO 1 STEP -1
  PRINT "Hallo"
NEXT x%
```

Anders als bei der *FOR-NEXT*-Konstruktion, kann mit *WHILE-WEND* die Abarbeitung einer Schleife von einer Bedingung abhängig gemacht werden.

Allgemeines Beispiel:

```
WHILE Bedingung
  ...
WEND
```

Die *WHILE-WEND*-Schleife wird so oft durchlaufen, solange die Bedingung erfüllt ist. Aber bedenken Sie, wenn die Bedingung bei der ersten Überprüfung nicht erfüllt ist, wird die Schleife **nicht** durchlaufen. Diese Möglichkeit haben Sie allerdings mit der *DO-LOOP*-Anweisung.

Beispiel:

```
WHILE x%<100
  PRINT "Hallo"
  x%=x%+1
WEND
```

Mit *DO-LOOP* verfügt Power Basic über eine der flexibelsten Kontrollstrukturen, die in den heutigen Programmiersprachen verfügbar sind. Um nicht zu verwirren, werden wir hier lediglich einige Beispielformulierungen anführen, die die Möglichkeiten dieses Konstruktes demonstrieren sollen.

Beispiel:

```
DO WHILE x%<100
  PRINT "Hallo"
  x%=x%+1
LOOP
```

Unser erstes Beispiel hat genau dieselbe Wirkung wie unser vorangehendes Beispiel zu der *WHILE-WEND*-Anweisung.

Beispiel:

```
DO
  PRINT "Hallo"
  x%=x%+1
LOOP WHILE x%<100
```

Mit unserem letzten Beispiel haben wir eine Schleifenkonstruktion vorgestellt, die mindestens einmal durchlaufen wird, da die Bedingung erst am Ende der Schleife abgefragt wird.

Neben den Schleifen sind an dieser Stelle noch die **Entscheidungskonstrukte** anzuführen. Hier stellt Power Basic die erweiterte *IF-THEN-ELSE/ END-IF-* und die *SELECT-CASE*-Anweisung zur Verfügung. Mit diesen Befehlen ist eine strukturierte Programmierung besonders einfach zu realisieren.

Allgemein:

```
IF 1.Bedingung THEN
     'Anweisungen, wenn 1.Bedingung wahr
  [ELSEIF 2.Bedingung THEN
     'Anweisungen, wenn 2.Bedingung wahr]
  [ELSE
     'Anweisungen für sonstige Fälle]
ENDIF
```

Die Teile der allgemeinen Formulierung, die in eckigen Klammern gesetzt sind, können auch entfallen, und die Anzahl der unterschiedlichen Bedingungen, die abgefragt werden, kann variieren. Trifft sowohl die erste als auch die zweite Bedingung zu, werden nur die Anweisungen für die erste Bedingung ausgeführt. In unserem folgenden Beispiel wird der Variablen *b%* ein Wert, abhängig von der Größe der Variablen *a%*, zugewiesen. Umfangreichere Anwendungen dieser Entscheidungskonstrukte werden wir im Verlaufe unseres Tutoriums noch kennenlernen.

Beispiel:

```
IF a%>1 AND a%<100 THEN
      b%=1        'b%=1, wenn 1< a% < 100
  ELSEIF a%<=1 THEN
      b%=0        'b%=0, wenn a% <= 1
  ELSE
      b%=-1       'b%=-1, wenn a% >= 100
END IF
```

Mit der *SELECT-CASE*-Anweisung kann der Programmablauf anhand eines Wertes gesteuert werden.

Allgemein:

```
SELECT CASE Ausdruck
  CASE 1.Prüfung
    'Anweisungen, wenn 1.Prüfung wahr
  [CASE 2.Prüfung
    'Anweisungen, wenn 2.Prüfung wahr]
  CASE ELSE
END SELECT
```

Beispiel:

```
SELECT CASE a%
  CASE 1
    'Anweisungen (wenn a%=1)
  CASE 3,4,5
    'Anweisungen (wenn a%=3 oder 4 oder 5)
  CASE 10 TO 20
    'Anweisungen (wenn 10 <= a% <= 20)
  CASE ELSE
    'Anweisungen (für sonstige Fälle)
END SELECT
```

Damit haben wir die wesentlichen Anweisungen zur Programmablaufsteuerung kennengelernt, die wir für die strukturierte Programmierung benötigen.

Unterprogramme und Funktionen

Über die Definition von Unterprogrammen und Funktionen haben wir die Möglichkeit, den Sprachumfang von Power Basic selbst zu erweitern. Mit Unterprogrammen lassen sich Befehle formulieren, die sich genauso verhalten, wie z.B. der Power-Basic-Befehl *CLS* zum Löschen des Bildschirms; mit *FUNCTION* können Power-Basic-Funktionen, wie z.B. SIN, COS, nachgebildet werden. Wir werden der Einfachheit halber die

allgemeine Formulierung an dieser Stelle vereinfachen. Umfangreiche Beispiele hierzu werden wir im Tutorium eingehend behandeln.

Benutzerdefiniertes Unterprogramm allgemein:

```
SUB Name [Parameterliste]
  'Anweisungen
END SUB
```

Beispiel:

```
SUB SchreibeHallo
  PRINT "Hallo"
END SUB

CALL SchreibeHallo
END
```

In unserem einfachen Beispiel wird lediglich durch ein benutzerdefiniertes Unterprogramm der Text *Hallo* auf den Bildschirm geschrieben. Der Aufruf dieser neuen Anweisung erfolgt mit *CALL*.

Benutzerdefinierte Funktion allgemein:

```
FUNCTION Name [Parameterliste]
  'Anweisungen
  Name = Rückgabewert
END FUNCTION
```

Beispiel:

```
FUNCTION Addition& (a%, b%)
  Addition& = a% + b%
END FUNCTION

PRINT Addition& (3, 7)
END
```

Ob Variablen nur innerhalb des Unterprogramms oder einer Funktion, oder aber global im gesamten Programm bekannt sind, kann über bestimmte Schlüsselwörter (*SHARED, LOCAL*) festgelegt werden. Standardmäßig sind alle in einem Unterprogramm oder einer Funktion eingeführten Variablen lokal, also nicht im gesamten Hauptprogramm bekannt. Ein Austausch von Variablenwerten ist allerdings auch über die Parameterlisten möglich.

Mit den Kenntnissen, die Sie bis jetzt erlangt haben, verfügen Sie bereits über einen umfangreichen Überblick über die Sprachsyntax von Power Basic. Um dieses Wissen zu schulen, aber auch um interne Befehle von Power Basic zu speziellen Programmierthemen kennenzulernen, sollten Sie sich nun mit dem Tutorium befassen. Bedenken Sie, daß man die Programmierung am besten anhand konkreter Beispiele erlernt.

Kapitel 2: Tutorium

2.1 Erste Sitzung: Das Konzept der Einnahmen-/Ausgabenverwaltung

Die Themen dieser Sitzung:

- Aufbau einer Benutzerbibliothek (Unit)
- Strukturierte Programmierung
- Modulare Programmierung
- Unterprogramme und Funktionen
- Basic-Eingabeanweisungen und Tastaturabfragen

Information

In der ersten Sitzung wollen wir uns mit der modularen und strukturierten Programmierung unter Power Basic vertraut machen, indem wir eine kleine Funktionsbibliothek anlegen. In dieser Benutzerbibliothek, einer sogenannten Unit, sind Funktionen und Unterprogramme enthalten, die universell einsetzbar sind. Diese Routinen werden wir in unserem Haushaltsprogramm nutzen, um eine professionell wirkende Oberfläche zu erstellen. Das Programm, das wir in unserem Tutorium entwickeln wollen, soll den

Namen HAUSHALT erhalten. Aufgabe ist es, eine einfache Einnahmen-/ Ausgabenüberschußrechnung zu programmieren, die eine Auswertung der in einer Datei abgelegten Daten, auf dem Bildschirm und dem Drucker erlaubt. Zur Veranschaulichung sollen die Ergebnisse auf dem Bildschirm auch grafisch darstellbar sein. Zur Optimierung des Programms wollen wir ferner eine einfache Hilfsfunktion und eine Fehlerbehandlungsroutine implementieren. Nun wollen wir uns aber zunächst mit der Oberflächenprogrammierung beschäftigen.

Als erstes wollen wir für unser Programm HAUSHALT eine Arbeitsfläche auf den Bildschirm zeichnen. Dieser Vorgang soll in einem Unterprogramm realisiert werden. Außerdem soll eine Kopfzeile mit dem Programmnamen und eine Fußzeile mit aktiven Sondertasten angezeigt werden.

Um die einzelnen Programmfunktionen, die in den späteren Sitzungen noch zu erstellen sind, aufrufen zu können, werden wir eine einfache Auswahlroutine formulieren. Die Auswahl soll in einem Fenster mit Hilfe der Cursorsteuerung erfolgen. Dazu brauchen wir eine einfache Routine zum Zeichnen eines Rahmens auf dem Bildschirm und eine Tastaturabfrage.

Abschließend benötigen wir für die Dateneingabe eine zu den internen Basic-Routinen verbesserte Eingaberoutine und eine Anweisung, die die Unterbrechung eines Programms erlaubt, bis eine beliebige Taste gedrückt wurde.

Um den Einsatz der Funktionsbibliothek zu demonstrieren, werden wir abschließend das Grundgerüst für unser Hauptprogramm erstellen.

Die Unit

Bei Power Basic handelt es sich um einen Compiler, der ein eigenständig ablauffähiges Maschinenprogramm aus einem Basic-Quelltext erstellen kann. Dieses wird entweder im Speicher direkt zur Ausführung gebracht, oder zunächst in eine Datei gespeichert, die später auch ohne Entwicklungsumgebung ausführbar ist. Bei großen Programmen ist bei jeder Quelltextänderung das gesamte Programm neu zu kompilieren. Um hier geringere Übersetzungszeiten zu erzielen, besteht die Möglichkeit, ausgetestete Routinen in eine Unit (Funktionsbibliothek) zu übernehmen. Nach der Generierung der Unit (Maschinencode) kann später bei Bedarf die

bereits übersetzte Bibliothek in ein Programm mit der *$LINK*-Anweisung eingebunden werden.

Jede Unit darf maximal 64 Kbyte Programmcode enthalten und besteht ausschließlich aus Unterprogrammen (*SUBs*) und Funktionen (*FUNCTIONs, DEF FNs*). Eine *$SEGMENT*-Anweisung ist innerhalb einer Unit nicht zulässig. Standardmäßig sind sämtliche Routinen einer Unit nicht öffentlich, d.h. außerhalb der Unit nicht bekannt (*PRIVATE*). *SUBs* und *FUNCTIONs* können außerhalb einer Benutzerbibliothek genutzt werden, wenn diese als *PUBLIC* (öffentlich) gekennzeichnet sind.

Strukturierte Programmierung

Power Basic stellt sämtliche Block- und Entscheidungsstrukturen zur Verfügung, die auch aus anderen höheren Programmiersprachen wie z.B. C oder Pascal bekannt sind.

- *SELECT-CASE*-Auswahl
- *WHILE-WEND*-Schleife
- *DO-LOOP*-Schleife
- *FOR-NEXT*-Schleife
- *IF-THEN-ELSE*-Abfrage

In Verbindung mit diesen Kontrollstrukturen, und der Möglichkeit der Erstellung von Unterprogrammen und Funktionen, können leicht überschaubare und sehr gut lesbare Quelltexte formuliert werden. Den Einsatz der obengenannten Anweisungen und die Möglichkeiten der strukturierten Programmierung werden wir anhand der Funktionsbibliothek genauer im Praxisteil besprechen.

Um strukturiert zu programmieren, sollten Sie häufiger benötigte Aufgaben innerhalb von Unterprogrammen und Funktionen formulieren. Verzichten Sie, wenn möglich, auf den Einsatz des *GOTO*-Befehls und auf Zeilennummern, die Sie vielleicht noch von den Basic-Interpretern her kennen. Auch sollten Sie globale Variablen nur in Maßen einsetzen, um den Überblick nicht zu verlieren. Variablen sollten immer mit aussagekräftigen Namen bezeichnet werden. Um Kontroll- und Entscheidungsstrukturen besser lesbar zu machen, sollten Sie mit Einrückungen arbeiten.

Modulare Programmierung

Power Basic ermöglicht über das Unit-Konzept und das Einbinden von Quelltexten mit der Compiler-Direktive *$INCLUDE* die modulare Programmierung. Ein Programm kann in der Regel in mehrere Teilbereiche (Module) aufgeteilt und getrennt voneinander entwickelt und ausgetestet werden. Dadurch sind Programme überschaubarer. Beachten Sie, daß Programme, die leicht lesbar sind, auch nach Monaten und Jahren noch ohne Schwierigkeiten gewartet werden können (Anpassungen an neue Hardware und Richtlinien, Programmerweiterungen etc.).

Unterprogramme und Funktionen

In Power Basic können Unterprogramme und Funktionen erstellt werden, mit denen Sie den Sprachumfang des Compilers nach Ihren eigenen Wünschen und Anforderungen erweitern können. Beispiele werden wir später im Praxisteil erläutern.

Bedenken Sie, daß eine Parameterliste eines Unterprogramms oder einer Funktion maximal 16 Elemente enthalten darf. Versuchen Sie nicht, diese unter allen Umständen auszunutzen. Lange Parameterlisten sind schlecht lesbar und können leicht zu Fehlern führen. Die Reihenfolge der Parameter sollte bei ähnlichem Aufbau in unterschiedlichen Routinen immer die gleiche sein. Benötigen Sie in zwei Routinen eine Farbe *vfarbe%* für den Vordergrund und eine Farbe *hfarbe%* für den Hintergrund, dann sollten in der einen als auch in der anderen Parameterliste die Farben immer in der gleichen Reihenfolge gesetzt werden, z.B. zuerst *vfarbe%*, dann *hfarbe%*.

Basic-Eingaberoutinen

Power Basic stellt mehrere Anweisungen zur Verfügung, die Eingaben durch einen Programmbenutzer ermöglichen. Diese sind innerhalb eines Programms mit einer Benutzeroberfläche allerdings nur beschränkt einsetzbar. Mit Hilfe der *INKEY$*-Anweisung läßt sich recht einfach eine eigene Eingaberoutine mit Power Basic realisieren.

LINE-INPUT und *INPUT* lassen keine Cursor-Kontrolle zu. Vorgegebene Eingabefelder werden damit ohne Kontrollmöglichkeiten überschrieben. Auch die *INPUT$*-Anweisung ist nur bedingt einsetzbar, da sie keine Tastaturcodes, die aus zwei Zeichen bestehen, erfaßt. Lediglich mit der *INKEY$*-Funktion sind umfassende Tastaturabfragen möglich.

Praxis

Wenn Power Basic von der Betriebssystemebene gestartet wird, meldet sich die integrierte Entwicklungsumgebung mit einem Eröffnungsbildschirm, wie in Bild 2.1.1 dargestellt. Standardmäßig ist als Arbeitsdatei NONAME.BAS voreingestellt. Wir wollen für unser Projekt aber zunächst den Namen WORK.BAS verwenden. Um diese Einstellung zu Beginn vorzunehmen, wählen wir *File → Write to* aus dem Menü.

In das geöffnete Eingabefeld tragen wir den gewünschten Namen ein und quittieren diesen mit [Return]. In der rechten oberen Bildschirmecke wird nun der von uns gewählte Dateiname erscheinen.

Überprüfen Sie, ob diese Datei nicht schon in Ihrem Verzeichnis vorhanden ist, da sie ohne Vorwarnung gelöscht wird, falls die Option »Warnung vor dem Überschreiben« im Menü *Options → Environment → Warn on overwrite* nicht auf *On* geschaltet ist. Ein Retten der Datei ist dann im nachhinein nicht mehr möglich.

Nun können wir über *Edit* in den Editor gelangen, wo wir den Quelltext eingeben und bearbeiten (editieren) werden.

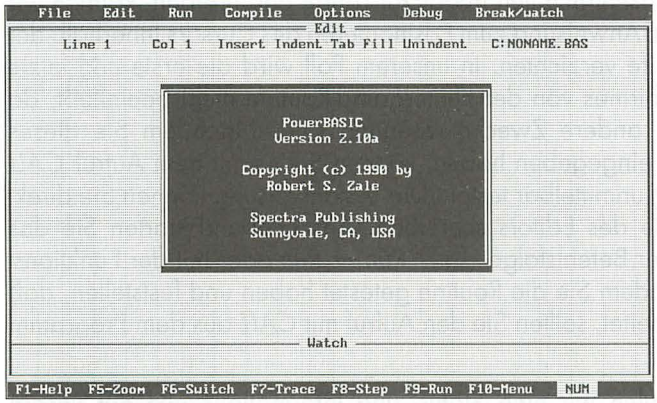

Bild 2.1.1: Power-Basic-Eröffnungsbildschirm

Wir definieren die Arbeitsfläche

Zunächst wollen wir die Arbeitsfläche für unser Programm zeichnen lassen. Dazu wählen wir ein Unterprogramm, an das später beim Aufruf Farbwerte für den Vorder- und Hintergrund übergeben werden. Da der gesamte Bildschirm als Arbeitsfläche dient, sind keine weiteren Parameter

erforderlich. Wir beschränken uns dabei auf den Textbildschirm mit 25 Zeilen und 80 Spalten.

```
SUB Hintergrund (vfarbe%, hfarbe%)
  COLOR vfarbe%, hfarbe%
  LOCATE 2, 1
  FOR x% = 1 TO 23
    PRINT STRING$(80, "▒");
  NEXT x%
END SUB
```

ALT drüchen 176 eingeben und Alt loslassen

Listing 2.1.1: Das Unterprogramm Hintergrund

Wir leiten ein Unterprogramm mit dem Schlüsselwort *SUB* ein, gefolgt von einem Namen und eventuell einer Parameterliste. Wie Sie an der Routine sehen, erfolgt der Bildschirmaufbau zeilenweise. Dabei haben wir die *FOR-NEXT*-Schleife verwendet. Als Hintergrundzeichen benutzen wir ein Grafikzeichen aus dem ASCII-Zeichensatz (vgl. Anhang). Beachten Sie, daß nach jeder *PRINT*-Anweisung automatisch ein Zeilenvorschub durchgeführt wird. In der 25. Zeile würde der Bildschirminhalt komplett um eine Zeile nach oben geschoben werden. Umgehen kann man dies, indem man die *PRINT*-Anweisung mit einem Semikolon abschließt. Dann wird der Zeilenvorschub unterdrückt. Und noch eine Anmerkung: Dadurch, daß wir sämtliche Bildschirmzeilen in ganzer Länge beschreiben, können wir auf eine *LOCATE*-Anweisung zum Positionieren des Cursors innerhalb der *FOR-NEXT*-Schleife verzichten. In unserem Fall wird die erste und letzte Zeile des Bildschirmes von der Hintergrund-Routine nicht beeinflußt, da diese Zeilen für andere Zwecke reserviert sind. Nachdem Sie dieses Unterprogramm eingegeben haben, können Sie es mit dem Aufruf *CALL Hintergrund(7,0)* unmittelbar testen. Die Testanweisung können Sie direkt in die erste Zeile des Editors schreiben. Anschließend starten Sie das Programm mit der Befehlsfolge [F10] *Run* → *Run* oder mit der Funktionstaste [F9]. Nachdem Sie die Routine getestet haben und feststellen, daß diese korrekt arbeitet, sollten Sie den Aufruf mit *CALL* aus dem Programm wieder entfernen.

Als nächstes wollen wir eine Status- und Informationszeile auf unsere Arbeitsfläche schreiben lassen. Da der Vorgang identisch ist, lediglich in einer anderen Bildschirmzeile stattfindet, bietet sich wieder die Formulierung als Unterprogramm an. Wir benötigen als Parameter eine Stringvariable, in der der Text für die Zeile übergeben wird, einen Farbparameter und einen Parameter für die zu verwendende Bildschirmzeile.

```
SUB InfoZeile (Text$, Zeile%, vfarbe%, hfarbe%)
  LOCATE Zeile%, 1
  COLOR vfarbe%, hfarbe%
  Text$ = SollLen$(Text$, 80)
  PRINT Text$;
END SUB
```

Listing 2.1.2: Das Unterprogramm InfoZeile

In diesem Unterprogramm wird lediglich der Cursor positioniert (*LOCATE*), die Farbeinstellung vorgenommen (*COLOR*) und der Text ausgegeben (*PRINT*). Zu erläutern bleibt allerdings die Textzuweisung. Dabei erfolgt der Aufruf einer benutzerdefinierten Funktion *SollLen$*, die sich ebenso verhält wie z.B. die Basic-interne Funktion *SIN*. Die Funktion gibt an die Variable *Text$* eine geänderte Variable *Text$* zurück. Bevor wir den Inhalt der Funktion besprechen, wollen wir zunächst den Quelltext vorstellen.

```
FUNCTION SollLen$ (Text$, Laenge%)
  TextLaenge% = LEN(Text$)
  IF TextLaenge% > Laenge% THEN
      Text$ = LEFT$(Text$, Laenge%)
    ELSEIF TextLaenge% < Laenge% THEN
      Text$ = Text$ + SPACE$(Laenge% - TextLaenge%)
  END IF
  SollLen$ = Text$
END FUNCTION
```

Listing 2.1.3: Die Funktion SollLen$

Die Funktion *SollLen$* erlaubt es, Strings auf eine vorgegebene Länge mit Leerzeichen aufzufüllen bzw. einen Text auf eine vorgegebene Länge zu kürzen. Damit sind bereits die beiden Funktionsparameter erklärt. Die Abfrage, ob eine Verkürzung oder eine Verlängerung erforderlich ist, erfolgt über eine *IF-THEN-ELSE*-Konstruktion. Das Ergebnis wird im Funktionsnamen direkt zurückgegeben, was die Zuweisung *SollLen$=Text$* erforderlich macht. Die Schlüsselwörter für eine Funktion sind *FUNCTION/END-FUNCTION*. Wir verwenden diese Funktion im Unterprogramm *InfoZeile*, um die Textzeile auf 80 Zeichen Länge aufzufüllen.

Fensterverwaltung einfach realisiert

Viele Programme arbeiten mit einer umfangreichen Fensterverwaltung. Eine einfache Realisierung ist über nur ein einziges Unterprogramm möglich, das wir an dieser Stelle vorstellen möchten. Der Programmaufbau ähnelt der Routine *Hintergrund*. Für den Fensterrahmen verwenden wir wieder Zeichen aus dem ASCII-Zeichensatz. Ein Rahmenaufbau soll auch für Bildschirmbereiche möglich sein. Daher benötigen wir Parameter für

die Fensterpositionierung und die Fenstergröße. Um das Unterprogramm möglichst überschaubar zu halten, verzichten wir auf Kontrollen.

```
SUB Fenster (Zeile%, Spalte%, Breite%, Hoehe%, vfarbe%, hfarbe%)
  COLOR vfarbe%, hfarbe%
  LOCATE Zeile%, Spalte%
  PRINT "╒" + STRING$(Breite%, "=") + "╕";
  FOR x% = Zeile% + 1 TO Zeile% + Hoehe%
    LOCATE x%, Spalte%
    PRINT "│" + SPACE$(Breite%) + "│";
  NEXT x%
  LOCATE x%, Spalte%
  PRINT "╘" + STRING$(Breite%, "=") + "╛";
END SUB
```

ALT + 205

Listing 2.1.4: Das Unterprogramm Fenster

Verwenden Sie aussagekräftige Variablennamen, ist ein Programm auch ohne eine Vielzahl von Kommentaren leicht lesbar. Ein auf den Bildschirm gezeichnetes Fenster muß später durch einen bestimmten Befehl auch wieder geschlossen werden. Hierzu schreiben wir die wohl einfachste aller Tataturabfragen, die ein Programm solange unterbricht, bis eine Taste gedrückt wurde.

```
SUB Warte
  WHILE INKEY$ = "": WEND
END SUB
```

Listing 2.1.5: Das Unterprogramm Warte

Nun haben wir bereits einige einfache Unterprogramme und eine Funktion vorgestellt, an denen man sehr gut die strukturierte Programmierung nachvollziehen kann. Die Befehle, die in diesen Programmeinheiten definiert sind, werden nicht unmittelbar ausgeführt. Ein Unterprogramm muß gesondert mit *CALL* und eine Funktion innerhalb einer Zuweisung aufgerufen werden.

Beispiel:

```
Unterprogramm:   CALL Warte
Funktion      :  Text$ = SollLen$ (Text$, 80)
```

Unterprogramme können direkt nach Fertigstellung durch einen Beispielaufruf im Hauptprogramm aufgerufen werden, so wie wir das bei der Routine *Hintergrund* vorgeführt haben. Von dieser Möglichkeit sollten Sie ausgiebigen Gebrauch machen. Erst wenn *SUBs* und *FUNCTIONs* korrekt zu arbeiten scheinen, ist die Übernahme in eine Unit sinnvoll. Versteckte Fehler oder falsche Ergebnisse sind aber auch dann noch möglich.

Programmaufbau

Wir haben die Routinen nacheinander, jeweils durch eine Leerzeile getrennt, in die Datei WORK.BAS eingegeben. Die Beispielaufrufe befinden sich am Beginn des Quelltextes und dienen als vorläufiges Hauptprogramm. Sie sollten versuchen, einen Quelltext nach einem einheitlichen Schema aufzubauen.

```
Deklarationsteil
Hauptprogramm
Unterprogramme und Funktionen
```

```
DECLARE SUB Warte()

CALL Warte

SUB Warte
  WHILE INKEY$="":WEND
END SUB
```

Auch bei längeren Programmen sollten Sie diesen Aufbau durchhalten. Über den Deklarationsteil kann Power Basic den korrekten Aufruf eines Unterprogramms oder einer Funktion überprüfen. Für die spätere Unit ist dieser Deklarationsteil unbedingt erforderlich. Bevor wir uns dem Hauptprogramm näher widmen, wollen wir zunächst zwei etwas anspruchsvollere Unterprogramme schreiben, die Sie sich etwas genauer ansehen sollten, da sie universell erweiterbar sind.

Eine allgemeine Auswahlroutine

Während eines Programmlaufes werden vom Benutzer immer wieder Eingaben, sogenannte Auswahlentscheidungen, gefordert. Kommerzielle Programme, so auch Power Basic, verwenden komplexe Menüstrukturen. Daß die Formulierung auch in Basic ohne große Probleme möglich ist, soll die nächste Routine beweisen.

```
SUB Auswahl(Text$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
  COLOR vfarbe%, hfarbe%     'Fensteraufbau
  Breite% = 10
  FOR x% = 1 TO Anzahl%      'max.Stringlänge
    IF LEN(Text$(x%)) > Breite% THEN Breite% = LEN(Text$(x%))
  NEXT x%
  FOR y% = 1 TO Anzahl%      'einheitl. Stringlänge
    Text$(y%) = SollLen$(Text$(y%), Breite%)
  NEXT y%
  Breite% = Breite% + 2      'Fenster
  CALL Fenster(5, 5, Breite%, Anzahl%, vfarbe%, hfarbe%)
  FOR x% = 1 TO Anzahl%
    IF x% = 1 THEN
```

```
      COLOR hfarbe%, vfarbe%
    ELSE
      COLOR vfarbe%, hfarbe%
  END IF
  LOCATE 5 + x%, 7, 0
  PRINT Text$(x%);
NEXT x%                          'Auswahl
BalkenPosition% = 1              'erster Eintrag aktiv
WHILE a$ <> CHR$(13)
  a$ = INKEY$
  IF LEN(a$) = 2 THEN
    AltPosition% = BalkenPosition%
    Taste% = ASC(RIGHT$(a$, 1))
    SELECT CASE Taste%
      CASE 72                    'Cursor Up
        IF BalkenPosition% = 1 THEN
            BalkenPosition% = Anzahl%
          ELSE
            BalkenPosition% = BalkenPosition% - 1
        END IF
      CASE 80                    'Cursor Down
        IF BalkenPosition% = Anzahl% THEN
            BalkenPosition% = 1
          ELSE
            BalkenPosition% = BalkenPosition% + 1
        END IF
      CASE ELSE
    END SELECT                   'Balken aktualisieren
    IF Taste% = 80 OR Taste% = 72 THEN
      COLOR vfarbe%, hfarbe% 'alter Balken
      LOCATE 5 + AltPosition%, 7, 0
      PRINT Text$(AltPosition%);
      COLOR hfarbe%, vfarbe% 'neuer Balken
      LOCATE 5 + BalkenPosition%, 7, 0
      PRINT Text$(BalkenPosition%);
    END IF
  END IF
  IF a$ <> CHR$(13) THEN a$ = ""
WEND
LOCATE , , 1
Wahl% = BalkenPosition%
END SUB
```

Listing 2.1.6: Die allgemeine Auswahlroutine

Wesentlicher Bestandteil dieser Routine ist die Tastaturabfrage, die durch die *INKEY$*-Anweisung ermöglicht wurde. Diese Funktion liefert einen Ein- oder Zwei-Zeichen-Tastaturcode (vgl. Anhang) zurück. Da wir die Cursorsteuerung behandeln wollen, benötigen wir einen Zwei-Zeichen-Code. In der Variablen *BalkenPosition%* merken wir uns den aktuell angewählten und in der Variablen *AltPosition%* den zuvor aktiven Menüpunkt. Mit Hilfe dieser Variablen können die Farbumstellungen vorgenommen werden.

Die Auswahl eines Menüpunkts erfolgt mit [Return]. Dann wird die *WHILE-WEND*-Schleife beendet. Damit eine Auswertung im Hauptprogramm möglich ist, wird die Wahl über die Variable *Wahl%* an das Hauptprogramm zurückgegeben. Ein vollständiges Verstehen dieser und der nächsten Routine ist noch nicht notwendig, da wir vor allem mit Hilfe dieser Unterprogramme eine Unit aufbauen wollen. Die Programmiertechnik wird in den weiteren Sitzungen noch vertiefend behandelt.

Eine einfache Eingaberoutine

Ebenso wie die Auswahlroutine arbeitet auch die Eingaberoutine. Hierbei wird allerdings ein Ein-Zeichen-Tastaturcode ausgewertet. Das Unterprogramm soll das Bearbeiten einer bestimmten Eingabe in einem genau definierten Bildschirmbereich zulassen. Dabei ist die [Backspace]-Taste als Löschtaste vorgesehen. Mit der [Return]-Taste wird die Eingaberoutine beendet. Die Parameterliste beinhaltet eine Vielzahl von Parametern. Drei davon wollen wir hier kurz erläutern. In der Variablen *Vorgabe$* wird eine Feldvorgabe (z.B.: "*Name:*"), in der Variablen *Laenge%* die Eingabefeldlänge übergeben. Wird die Variable *Eingab$* vor dem Unterprogramm bereits gesetzt (die Länge muß kleiner gleich der Länge der Eingabefeldlänge sein), kann ein Inhalt in das Eingabefeld vorgegeben werden. Über diesen Parameter wird aber auch die bearbeitete Eingabe zurückgegeben. Den Einsatz dieser Routine werden wir in späteren Lektionen noch genauer erläutern. Eine Erweiterung der Eingaberoutine werden wir für die spätere Dateneingabe vornehmen.

```
SUB Eingabe (Vorgabe$, Eingab$, Laenge%, Zeile%, Spalte%,_
             vfarbe%, hfarbe%)
  LOCATE Zeile%, Spalte%
  COLOR vfarbe%, hfarbe%
  PRINT Vorgabe$;
  COLOR hfarbe%, vfarbe%
  PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
  MinSpalte% = LEN(Vorgabe$) + Spalte%
  MaxSpalte% = LEN(Vorgabe$) + Spalte% + Laenge% - 1
  EditSpalte% = LEN(Vorgabe$) + Spalte% + LEN(Eingab$)
  IF EditSpalte%>MaxSpalte% THEN EditSpalte=EditSpalte%-1
  WHILE Zeichen$ <> CHR$(13)
    LOCATE Zeile%, EditSpalte%, 1
    Zeichen$ = INKEY$
    IF LEN(Zeichen$) = 1 THEN
      SELECT CASE ASC(Zeichen$)
        CASE 8   'Backspace
          IF EditSpalte% > MinSpalte% THEN
            IF LEN(Eingab$) = Laenge% THEN
              EditSpalte% = EditSpalte% + 1
```

```
            END IF
            Eingab$ = LEFT$(Eingab$, LEN(Eingab$) - 1)
            LOCATE Zeile%, MinSpalte%
            PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
            IF MinSpalte% < EditSpalte% THEN
              EditSpalte% = EditSpalte% - 1
            END IF
          END IF
        CASE 32 TO 254
          IF EditSpalte% <= MaxSpalte% THEN
            IF LEN(Eingab$) = Laenge% THEN
              Eingab$ = LEFT$(Eingab$, LEN(Eingab$) - 1)
            END IF
            Eingab$ = Eingab$ + Zeichen$
            LOCATE Zeile%, MinSpalte%
            PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
            IF MaxSpalte% > EditSpalte% THEN
              EditSpalte% = EditSpalte% + 1
            END IF
          END IF
        CASE ELSE
      END SELECT
    END IF
    IF Zeichen$ <> CHR$(13) THEN Zeichen$ = ""
  WEND
END SUB
```

Listing 2.1.7: Eine allgemeine Eingaberoutine

Beachten Sie, daß die Feldvorgabe und das Eingabefeld zusammengehören. Die Positionierung bezieht sich daher immer auf die Feldvorgabe und das Eingabefeld wird daran direkt angehängt. Die Gesamtlänge dieser Einheit darf in dieser Programmfassung maximal eine Bildschirmzeile (80 Zeichen) betragen.

Das Hauptprogramm

Damit wir die Lauffähigkeit unserer Routinen gleich testen können, wollen wir an dieser Stelle das Programmgerüst für unser Hauptprogramm entwickeln. Beachten Sie wieder den gewählten Programmaufbau von Deklarationsteil, Hauptprogrammteil und Funktionsbibliotheksteil. Der Programmablauf wird durch unsere Auswahlroutine gesteuert, indem der zurückgegebene Wert für den gewählten Menüpunkt in einer *SELECT-CASE*-Abfrage verarbeitet wird.

```
DECLARE SUB InfoZeile (STRING, INTEGER, INTEGER, INTEGER)
DECLARE FUNCTION SollLen$ (STRING, INTEGER)
DECLARE SUB Hintergrund (INTEGER, INTEGER)
DECLARE SUB Eingabe (STRING, STRING, INTEGER, INTEGER, INTEGER, INTEGER,_
            INTEGER)
```

```
DECLARE SUB Fenster (INTEGER, INTEGER, INTEGER, INTEGER, INTEGER, INTEGER)
DECLARE SUB Auswahl (STRING ARRAY, INTEGER, INTEGER, INTEGER, INTEGER)


$STACK 4000                    'Stapelspeicher
CLS                            'Hauptprogramm HAUSHALT
vfarbe% = 7                    'Daten initialisieren
hfarbe% = 1
Anzahl% = 8                    'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
Menue$(3) = "Datenauswertung"
Menue$(4) = "Druck"
Menue$(5) = "Grafik"
Menue$(6) = "OS-Shell"
Menue$(7) = "Hilfe"
Menue$(8) = "Ende"

CALL Hintergrund(vfarbe%, hfarbe%)
WHILE Wahl%<>8
  CALL InfoZeile("HAUSHALT Version 1.00 - Einnahmen-"+_
                 "/Ausgabenüberschußrechnung", 1, 7, 1)
  CALL InfoZeile(CHR$(24)+CHR$(25)+": Anwahl Menüeintrag
  RETURN:"+_
                 "Menüauswahl", 25, vfarbe%, hfarbe%)
  CALL Auswahl(Menue$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
  SELECT CASE Wahl%
    CASE 1
      'hier wird später die Arbeitsdatei gewählt
    CASE 2
      'hier wird später die Dateneingabe vorgenommen
    CASE 3
      'hier wird später die Datenauswertung durchgeführt
    CASE 4
      'hier werden später die Ergebnisse ausgedruckt
    CASE 5
      'hier wird später die Grafik ausgegeben
    CASE 6
      'Schnittstelle zum Betriebssystem
    CASE 7
      'hier wird später die Hilfsfunktion integriert
    CASE ELSE
  END SELECT
WEND
COLOR 7, 0
CLS
PRINT "Programm beendet..."
PRINT "Fenster- und Eingaberoutine-Demonstration:"
CALL Fenster(5, 5, 70, 10, vfarbe%, hfarbe%)
Eingab$ = "Test"
CALL Eingabe("Bitte etwas eingeben: ", Eingab$, 20, 6, 6,
           vfarbe%,_hfarbe%)
LOCATE 10, 10
COLOR 7,0
```

```
CLS
PRINT "Sie haben >" + Eingab$ + "< eingegeben..."
PRINT "DEMO beendet..."
END

'an dieser Stelle folgen die allgemein einsetzbaren Funktionen
'und Unterprogramme
'auf das Listing dieser Routinen soll an dieser Stelle verzichtet werden
```

Listing 2.1.8: Die erste Fassung unseres Hauptprogramms

Dieses Programm befindet sich nun in der Gesamtheit im Speicher. Sie sollten diese Datei regelmäßig, spätestens aber vor jedem Programmstart über das Menü *File* → *Save* auf Diskette bzw. Festplatte sichern. Wenn Sie die Datei WORK.BAS komplett in den Editor eingegeben haben, ergibt sich ein Bildschirm, wie in Bild 2.1.2 dargestellt.

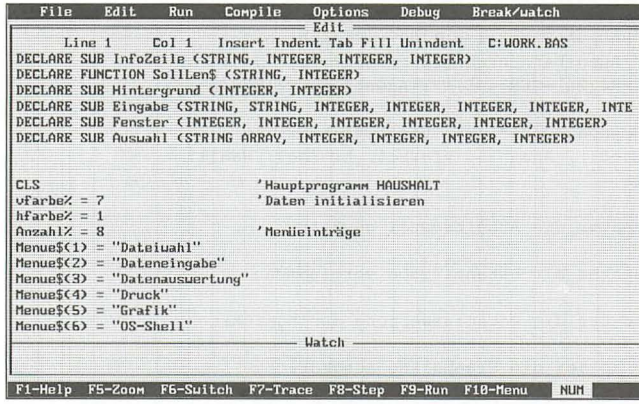

Bild 2.1.2: Erstellung der allgemeinen Routinen

Unser Programm kann jetzt direkt ausgetestet werden, indem Sie es im Speicher in Maschinensprache übersetzen und ausführen lassen. Dazu verlassen Sie den Editor über die Funktionstaste [F10] und wählen den Menüpunkt *Run* → *Run* (Bild 2.1.3). Das Programm wird unmittelbar ausgeführt. Sollten Sie den Quelltext fehlerhaft eingegeben haben (z.B. Tippfehler), so wird der Compiler mit entsprechenden Fehlermeldungen abbrechen. Diese Fehler müssen Sie korrigieren, indem Sie in den Editor zurückgehen und den Quelltext editieren.

```
File   Edit   Run   Compile   Options   Debug   Break/watch
Line 1   C:                  Run              F9   Fill Unindent   C:WORK.BAS
DECLARE SUB InfoZe           Program reset Ctrl-F2 EGER, INTEGER)
DECLARE FUNCTION S           Go to cursor     F4
DECLARE SUB Hinter           Trace into       F7
DECLARE SUB Eingab           Step over        F8   R, INTEGER, INTEGER, INTEGER, INTE
DECLARE SUB Fenste           User screen   Alt-F5 GER, INTEGER, INTEGER, INTEGER)
DECLARE SUB Auswah           Block run             INTEGER, INTEGER, INTEGER)

CLS                                  'Hauptprogramm HAUSHALT
vfarbe% = 7                          'Daten initialisieren
hfarbe% = 1
Anzahl% = 8                          'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
Menue$(3) = "Datenauswertung"
Menue$(4) = "Druck"
Menue$(5) = "Grafik"
Menue$(6) = "OS-Shell"
                                   Watch
F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Run F10-Menu   NUM
```

Bild 2.1.3: Testlauf der Bibliotheksfunktionen

Schreiben einer Unit

Nun haben wir die Grundlagen gelegt, um eine Unit zu schreiben. Wir haben bereits im Informationsteil erwähnt, daß nur Unterprogramme und Funktionen in einer Unit enthalten sein dürfen. Wir müssen also die Datei WORK.BAS in mehrere Teildateien aufsplitten. Dies gestaltet sich über den Editor recht einfach. Die Unit soll den Namen TOOLS.BAS erhalten. Den Deklarationsteil werden wir gesondert in die Datei TOOLS.H, das Hauptprogramm in die Datei WORK2.BAS speichern. Der ursprüngliche Quelltext soll dabei als Sicherungsdatei erhalten bleiben. Um diese Arbeitsschritte durchführen zu können, werden wir mit den Blockbefehlen des Editors arbeiten. Diese sind nachfolgend kurz aufgelistet:

Blockanfang markieren: [Strg] + [K] [B]
Blockende markieren: [Strg] + [K] [K]
Block in Datei speichern: [Strg] + [K] [W]

Gehen Sie in den Editor, um die Datei WORK.BAS aufzuteilen. Positionieren Sie den Cursor zunächst auf den Anfang der ersten Zeile des Deklarationsteils und drücken Sie die Tastenkombination [Strg] + [K] [B]. Dann bewegen Sie den Cursor auf das Ende des Deklarationsteiles und drücken [Strg] + [K] [K]. Sie werden sehen, daß der markierte Textbereich farblich hervorgehoben wird. Mit [Strg] + [K] [W] können Sie nun diesen Block in eine Datei speichern. Geben Sie bei der Dateinamensabfrage TOOLS.H ein. Ebenso verfahren Sie mit dem Hauptprogrammteil und dem Unit-Teil. Nun haben Sie drei zusätzliche Dateien erzeugt, die wir gesondert behandeln werden.

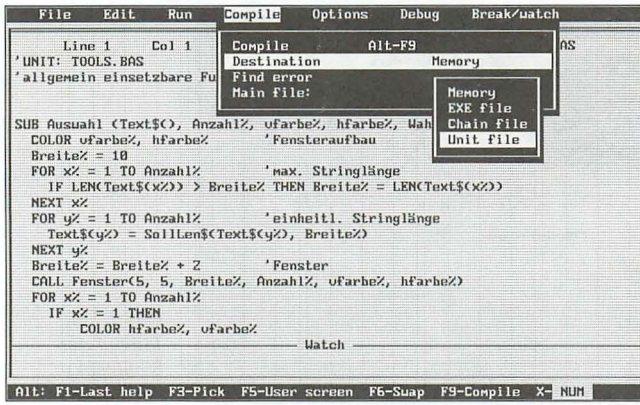

Bild 2.1.4: Erstellung einer Unit

Laden Sie die Unit mit dem Namen TOOLS.BAS mit Hilfe der Funktionstaste [F3] in den Speicher. Im Editor müssen wir an das Ende jeder Parameterzeile das Schlüsselwort *PUBLIC* einfügen, damit sämtliche Routinen später von unserem Hauptprogramm genutzt werden können. Danach verlassen wir den Editor mit [F10] und wählen das Menü *Compile → Destination → Unit*. Damit geben wir dem Compiler bekannt, daß er bei der nächsten Übersetzung ein Bibliotheksmodul erzeugen soll (Bild 2.1.4). Anschließend lassen wir die Unit in Maschinensprache übersetzen. Dazu brauchen Sie lediglich die Tastenkombination [Alt] + [F9] betätigen.

Auf der Festplatte befinden sich mittlerweile folgende Dateien:

WORK.BAS	Quelldatei gesamt
WORK2.BAS	Grundgerüst Hauptprogramm
TOOLS.BAS	Quelltexte der Unterprogramme
TOOLS.H	Deklarationsteil
TOOLS.PBU	kompilierte Unit

An dieser Stelle müssen wir das Hauptprogramm an das Unit-Konzept anpassen. Dazu laden wir zunächst die Datei WORK2.BAS mit der Funktionstaste [F3]. Das Hauptprogramm wollen wir zu einem eigenständig ablauffähigen Programm übersetzen. Deshalb wählen wir das Menü *Compile → Destination → EXE file* (Bild 2.1.5). Beachten Sie, daß unsere Umstellung auf Unit ansonsten weiterhin gültig wäre.

Wie Sie in der nächsten Abbildung erkennen können, haben wir unser Hauptprogramm mit Kommentarzeilen versehen, die eine Identifizierung der Quelldatei auch später noch möglich machen.

```
File    Edit    Run    Compile    Options    Debug    Break/watch
                                  Edit
    Line 14    Col 1    Insert Indent Tab Fill Unindent * C:HAUSHALT.BAS
$INCLUDE "TOOLS.H"              'Deklarationen der Unit TOOLS
$LINK "TOOLS.PBU"               'Benutzerbibliothek TOOLS einbinden

'*************************************************************************
'* Programmname              : HAUSHALT.BAS Version 1.00                 *
'* Compiler                  : Power BASIC 2.10                          *
'* verwendete Headerdateien: TOOLS.H                                     *
'* verwendete Units          : TOOLS.PBU                                 *
'* geschrieben               : April/Mai 1991                            *
'* letzte Änderung           : 16.05.1991                                *
'*************************************************************************

CLS                             'Hauptprogramm HAUSHALT
vfarbe% = 7                     'Daten initialisieren
hfarbe% = 1
Anzahl% = 8                     'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
                                  Watch

F1-Help  F5-Zoom  F6-Switch  F7-Trace  F8-Step  F9-Run  F10-Menu   NUM
```

Bild 2.1.5: Erstellung eines eigenständig ablauffähigen Demonstrationsprogramms

Dieser Aufbau kann Ihnen als Beispiel dienen, daher sind die Zeilen auf der nächsten Seite noch einmal abgedruckt. Versuchen Sie zu Beginn Ihres Quelltextes einen ausführlichen Kommentar zu formulieren, der alle wichtigen Informationen zum Programm enthalten sollte. Außerdem sind zwei Compiler-Befehle eingefügt, die ebenfalls genauer zu erläutern sind.

Compiler-Befehle:

```
$INCLUDE "TOOLS.H"          'Deklarationen der Unit TOOLS
$LINK "TOOLS.PBU"           'Benutzerbibliothek TOOLS einbinden
```

Mit dem ersten Befehl binden wir die Datei TOOLS.H, eine sogenannte Headerdatei, in unser Programm ein. Damit werden dem Compiler die Routinen und ihre Syntax in der Unit TOOLS bekannt gemacht. Die *$LINK*-Anweisung teilt dem Compiler mit, in das Maschinenprogramm die Unit TOOLS.PBU mit einzubinden. Erst mit diesen Befehlen kann das Hauptprogramm direkt ausgeführt werden.

Zuvor haben wir bereits einen anderen Compiler-Befehl in unserem Programm verwendet, den wir in diesem Zusammenhang ebenfalls erläutern wollen. Durch den Einsatz der Anweisung *$STACK* 4000 läßt sich der Stapelspeicher von standardmäßig 1536 Byte auf 4000 Byte erhöhen. In diesem Speicherbereich werden Rücksprungadressen, Parameterübergaben und lokale Variablen gespeichert. Da wir die Unterprogrammtechnik ausführlich nutzen wollen, würde der bereitgestellte Stapelspeicher für unser endgültiges Programm nicht unbedingt ausreichen. Ist das Menü *Options* → *Compiler* → *Error tests* → *Stack test* auf *ON* gesetzt, meldet Power Basic, wenn der Stapelspeicher für das aktive Programm

nicht ausreicht. Die *$STACK*-Anweisung darf innerhalb eines Programms nur einmal verwendet werden.

Kommentarzeilen des Hauptprogramms:

```
'*********************************************************************
'* Programmname              : HAUSHALT.BAS                          *
'* Compiler                  : Power Basic 2.10                      *
'* verwendete Headerdateien: TOOLS.H                                 *
'* verwendete Units          : TOOLS.PBU                             *
'* geschrieben               : April/Mai 1991                        *
'* letzte Änderung           : 27.04.1991                            *
'*********************************************************************
```

Neben der Kommentierung Ihrer Hauptprogramme sollten Sie auch Header-Dateien und Quelltexte zu Units ausreichend kommentieren.

Bild 2.1.6: Der Eröffnungsbildschirm unseres Programms Haushalt

Jetzt können wir mit [Alt] + [F9] die EXE-Datei erzeugen lassen. Auf der Festplatte wird die Datei WORKS2.EXE gespeichert, die nun direkt von der Kommandozeilenebene aus gestartet werden kann. Aber auch eine Ausführung innerhalb der Benutzeroberfläche ist möglich. Dazu sollten Sie nach [Alt] + [F9] die Menüfolge *Run* → *Run* eingeben.

Den Eröffnungsbildschirm unseres Programms HAUSHALT können wir in Bild 2.1.6 betrachten. Sie sehen, daß man mit sehr wenig Aufwand ein ansprechendes Programm realisieren kann.

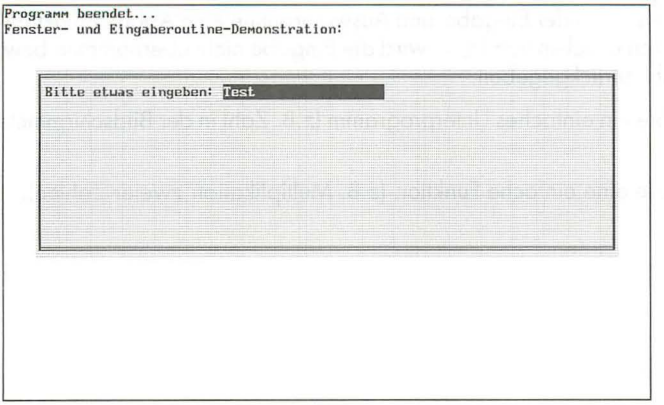

Bild 2.1.7: Testlauf der Eingabe- und Fensterroutine

An das eigentliche Hauptprogramm (s.o.) haben wir eine kleine Demonstration, der an dieser Stelle noch nicht benötigten Routinen angehängt. Die Bildschirmausgabe dieser Zeilen ist in Bild 2.1.7 dargestellt.

Die Grundlagen der strukturierten und modularen Programmierung sind damit gelegt, und wir können uns in den nächsten Sitzungen gezielt mit unserem Projekt HAUSHALT befassen.

Workshop

Checkliste

1. Was versteht man unter »strukturierter Programmierung«?
2. Was ist das »Unit-Konzept«?
3. Was ist »modulare Programmierung«?
4. Was bedeutet das Schlüsselwort »PUBLIC«?
5. Was ist ein »Unterprogramm«, was eine »Funktion«?
6. Nennen Sie die wichtigsten Block- und Entscheidungsstrukturen in Power Basic!

Ideen für eigene Übungen

- Erweitern Sie die Tastaturabfrage der Eingaberoutine so, daß auch eine Cursorsteuerung zum Bewegen des Textcursors aktiv ist (Zwei-Zeichen-Tastaturcode).
- Erweitern Sie die Auswahlroutine so, daß auch Menüeinträge durch das Eintasten ihrer Anfangsbuchstaben angesteuert werden können.

- Implementieren Sie in der Eingabe- und Auswahlroutine eine Abbruchfunktion. Das heißt, durch Drücken von Esc wird die Eingabe nicht übernommen bzw. keine Auswahl zurückgegeben.
- Formulieren Sie ein einfaches Unterprogramm (z.B. Zahl in der Bildschirmmitte ausgeben).
- Formulieren Sie eine einfache Funktion (z.B. Multiplikation zweier Zahlen).

2.2 Zweite Sitzung: Wahl einer Arbeitsdatei für Buchungssätze

Die Themen dieser Sitzung:

- DOS-Befehle von Power Basic
- Umgang mit Dateien
- Dateiarten
- Dateinamensvergabe unter Power Basic
- Dateiverzeichnisse ermitteln
- Verwaltung von Datenfeldern

Information

Haben wir uns in der ersten Sitzung mit der Oberflächenprogrammierung unseres Programms HAUSHALT befaßt, werden wir nun die Schnittstelle von Power Basic zum Betriebssystem ausnutzen, um eine Arbeitsdatei aus dem Dateiverzeichnis auszuwählen. Dazu wollen wir die Dateinamen, die unser Programm verarbeiten kann, in ein Datenfeld (Array) einlesen und an unsere allgemeine Auswahlroutine aus unserer ersten Sitzung übergeben. Die einzelnen Dateieinträge sollen dabei alphabetisch sortiert zur Auswahl angeboten werden, so wie es in den meisten kommerziellen Programmen auch der Fall ist. Da unter Umständen keine Datei im aktuellen Verzeichnis verarbeitbar ist, muß eine Möglichkeit geschaffen werden, um neue Dateien komfortabel anlegen zu können. Welche Arbeitsdatei gewählt ist, soll in unserer Fußleiste auf dem Bildschirm mit eingetragen werden. Um die Programmierung einfacher zu gestalten, wird für unser Programm immer nur eine einzige Arbeitsdatei aktiv sein. Auf Dateiverknüpfungen werden wir bewußt verzichten.

Wäre die Lösung dieser Aufgaben mit anderen Basic-Dialekten nur mit viel Aufwand zu realisieren, gestaltet sich die Programmierung in Power Basic durch den stark erweiterten Sprachumfang sehr einfach. So sind bereits alle benötigten Befehle zum Behandeln von Verzeichnissen und Dateien sowie Routinen zur Listenverwaltung vorhanden.

DOS-Befehle von Power Basic

Beim Umgang mit Dateien ist eine Programmiersprache immer vom Betriebssystem abhängig, das für die Dateiorganisation und die Verzeichnisstruktur verantwortlich ist. Die wichtigsten DOS-Befehle, die bereits von Power Basic zur Verfügung gestellt werden, sind in Tabelle 2.2.1 zusammengestellt. Nähere Informationen zu den einzelnen Anweisungen erhalten Sie über die interne Hilfsfunktion, wie in Kapitel 1 beschrieben.

ATTRIB-Funktion	Dateiattribute ermitteln
ATTRIB-Befehl	Dateiattribute setzen
CHDIR	Verzeichnis wechseln
CHDRIVE	Laufwerk wechseln
CURDIR$	aktuelles Verzeichnis ermitteln
DIR$	Dateiverzeichnisse ermitteln
FILES	Inhaltsverzeichnis anzeigen lassen
KILL	Datei löschen
MKDIR	Verzeichnis anlegen
NAME	Datei umbenennen
RMDIR	leeres Verzeichnis löschen

Tabelle 2.2.1: Wichtige DOS-Anweisungen von Power Basic

Power Basic verfügt noch über weitere Anweisungen für das Betriebssystem, die wir an dieser Stelle allerdings nicht erläutern wollen. Sie sehen, daß Sie umfangreiche Befehle zur Verfügung gestellt bekommen, die es Ihnen ermöglichen, einen Dateimanager für Ihre eigenen Programme zu entwickeln, so daß ein Rücksprung aus Ihrem Programm zur Kommandozeilenebene des Betriebssystems in der Regel überflüssig wird. Notwendig für den Einsatz dieser Befehle ist, daß Sie umfangreiche Kenntnisse über die Verzeichnisstruktur und die Dateiverwaltung des Betriebssystems besitzen.

Umgang mit Dateien

Da wir in unserem Programm mit der gewählten Datei arbeiten, d.h. Daten aus der Datei lesen bzw. Daten in die Datei schreiben wollen, müssen wir die Datei zunächst öffnen. Ist eine Bearbeitung der Datei nicht mehr nötig, muß diese dementsprechend wieder geschlossen werden. Die Arbeitsdatei, die wir mit unserer Auswahlroutine anwählen, kann direkt nach der Anwahl durch das Schlüsselwort *OPEN* geöffnet werden. Das Schließen der Datei kann am Programmende mit *CLOSE* geschehen. Dabei ist zu

beachten, daß einer Datei beim Öffnen immer eine bestimmte Dateinummer zugewiesen wird. Es dürfen niemals zwei Dateien mit derselben Dateinummer geöffnet werden, da ein Programm dann mit einer Laufzeitfehlermeldung abbricht. Wählen wir in unserem Programm HAUSHALT also eine neue Arbeitsdatei, muß die alte Datei unbedingt zuerst geschlossen werden, damit wir wieder dieselbe Dateinummer verwenden können.

Dateiarten

Beim Öffnen einer Datei muß angegeben werden, ob die Datei als sequentielle, als Direktzugriffsdatei oder als Binärdatei geöffnet werden soll. Eine sequentielle Datei ist eine Datei, bei der alle Lese- und Schreibvorgänge immer am Dateianfang beginnen müssen. Wollen Sie zum Beispiel den letzten von 5000 Datensätzen bearbeiten, müssen Sie zuvor auch die anderen 4999 Datensätze lesen. Ein Hauptanwendungsgebiet ist die Textverarbeitung. Die meisten Texteditoren (auch der Editor von Power Basic) speichern ihre Texte in sequentiellen Dateien. Direktzugriffs- oder Randomdateien werden insbesondere im Bereich der Dateiverwaltung eingesetzt. Hier kann gezielt auf jeden Satz über eine Satznummer zugegriffen werden. Wollen Sie in einer Randomdatei z.B. den 5000. Datensatz verarbeiten, so brauchen Sie nicht alle anderen Datensätze auch zu lesen. Dadurch haben Sie einen erheblichen Geschwindigkeitsvorteil bei der Programmausführung gegenüber den sequentiellen Dateien. Jeder Datensatz in einer Direktzugriffsdatei hat, anders als in einer sequentiellen Datei, immer eine einheitliche Länge. Für die Dateiverwaltung in unserem Haushaltsprogramm werden wir diesen Dateityp verwenden. Außerdem bietet Power Basic den Umgang mit Dateien im Binärmodus an. Dateiinformationen lassen sich hier byteweise (zeichenweise) lesen oder schreiben. Insbesondere zur Verarbeitung von Maschinenprogrammen bietet sich dieser Dateimodus an.

Syntax:

```
OPEN Dateiname FOR Modus AS Nummer
```

```
Modus : sequentiell schreiben: OUTPUT
        sequentiell lesen    : INPUT
        sequentiell anhängen : APPEND
        Direktzugriff        : RANDOM
        binär                : BINARY
```

Dateiverzeichnisse ermitteln

Power Basic stellt neben der *FILES*-Anweisung, die es ähnlich dem DOS-DIR/W-Befehl ermöglicht, den Inhalt des aktuellen Dateiverzeichnisses anzuzeigen, auch eine spezielle Funktion mit dem Namen *DIR$* zur Verfügung. Ermöglicht *FILES* lediglich eine Anzeige der Dateinamen auf dem Bildschirm, so können mit der Funktion *DIR$* Dateinamen nach einem bestimmten Muster gesucht und bestimmten Variablen übergeben werden. Die Möglichkeiten dieser Funktion werden wir in unserem Praxisteil demonstrieren.

Verwaltung von Datenfeldern

Power Basic beinhaltet einige sehr leistungsfähige Anweisungen zur Listenverwaltung. Stellen Sie sich vor, Sie wollen in Ihrem Programm eine Liste mit 500 Namen erstellen. Dazu haben Sie ein Datenfeld mit dem Namen *Namen$* und der Anweisung *DIM Namen$(1:500)* dimensioniert. Im Anschluß daran haben Sie die einzelnen Elemente initialisiert, so daß jedem Element ein bestimmter Name zugewiesen ist, wie in der nachfolgenden Tabelle dargestellt.

```
Namen$(1)   = "ANTON"
Namen$(2)   = "EGON"
Namen$(3)   = "KAUS"
Namen$(4)   = "GUNDA"
Namen$(5)   = "INGO"
Namen$(6)   = "GERO"
Namen$(7)   = "PIA"
Namen$(8)   = "MARITA"
Namen$(9)   = "DOMINIK"
...
...
Namen$(500) = "ANDREAS"
```

Wenn Sie diese Liste nun alphabetisch aufsteigend oder absteigend sortieren lassen wollen, oder nach bestimmten Elementen suchen, löschen und einfügen wollen, so brauchen Sie lediglich die *ARRAY*-Anweisung von Power Basic zu verwenden, die die Listenverwaltung von sämtlichen in Power Basic zur Verfügung stehenden Datentypen erlaubt. Die vereinfachte Syntax der *ARRAY*-Anweisung hat folgenden Aufbau:

```
ARRAY Schlüssel Datenfeld Optionen/Ausdruck
```

```
Schlüssel: SORT          - Datenfeld sortieren
           SCAN          - Datenfeld durchsuchen
           INSERT        - Element einfügen
           DELETE        - Element löschen

Datenfeld: N$()          - gesamtes Datenfeld
           N$() FOR 10   - Element 1 bis 10
           N$(5) FOR 10  - Element 5 bis 10

Optionen : ASCEND        - aufsteigend sortieren
           DESCEND       - absteigend sortieren
           COLLATE       - spezielle Stringsortierung
```

Wir werden diese Anweisung in unserer Dateiauswahlroutine zum Sortieren der Dateiliste verwenden.

Namensvergabe unter MS-DOS

Dateinamen, die von einer Programmiersprache vergeben werden können, müssen immer auch zulässige Dateinamen für das Betriebssystem, in diesem Fall MS-DOS oder dazu kompatiblen Betriebssystemen, wie z.B. DR-DOS, sein. Jeder Dateiname kann aus bis zu 8 Zeichen bestehen, gefolgt von einem Punkt und einem sogenannten Suffix bzw. Dateikürzel. Für die Dateikürzel haben sich bereits eine Vielzahl von Kombinationen eingebürgert. Die wichtigsten, an die wir uns für die Basic-Programmierung halten sollten und einige Beispielnamen, sind in der nachfolgenden Liste aufgeführt.

Dateikürzel:

.BAS	Basic-Quelldatei
.H	Headerdatei
.INC	INCLUDE-Datei
.PBU	Power-Basic-Unit
.EXE	ausführbares Programm
.TXT	Textdatei
.DAT	Datendatei
.DBF	Datenbank-Datei

Beispiele:

HAUSHALT.BAS	Quelltext Haushaltsprogramm
TOOLS.BAS	Quelltext der Unit TOOLS
TOOLS.H	Headerdatei zur Unit TOOLS

TOOLS.PBU	Funktionsbibliothek TOOLS
HAUSHALT.DAT	Daten-Datei für HAUSHALT.EXE
HAUSHALT.EXE	ausführbares Programm HAUSHALT

An diese Richtlinien der Namensvergabe werden wir uns auch in unserem Programm halten.

Praxis

Wie Sie sicherlich gemerkt haben, wurden in unserer ersten Sitzung eine Sammlung von Unterprogrammen und Funktionen angelegt, die wir in allen Programmen einsetzen können. Das Hauptprogramm bekam von uns den Namen WORK2.BAS und war zunächst in erster Linie als Demonstrations- und Testprogramm für eine Funktionsbibliothek angelegt. Die Grundstruktur für unser Haushaltsprogramm haben wir allerdings bereits in dieses Programm eingebaut. Da wir das Demo-Programm als unveränderte Kopie erhalten wollen, legen wir uns zunächst unser Hauptprogramm HAUSHALT an.

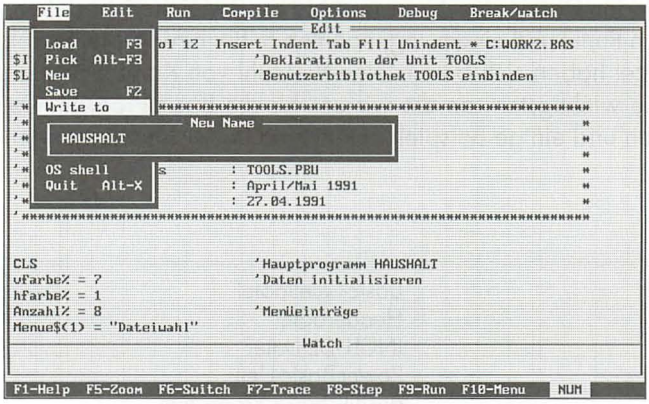

Bild 2.2.1: Erstellung von HAUSHALT.BAS

Im Editor befindet sich noch unser Programm WORK2.BAS. Um eine Kopie für unser Hauptprogramm zu erzeugen, gehen wir mit [F10] in das Menü und wählen die Menüfolge *File → Write To*. In das sich anschließend öffnende Eingabefenster tragen wir nun den Namen HAUSHALT ein und quittieren mit [Return] (Bild 2.2.1). Da wir kein Suffix eingegeben haben, wird hier automatisch die Endung .BAS angehängt. Nach dieser Eingabe befindet sich neben WORK2.BAS die Kopie HAUSHALT.BAS auf der Festplatte. Die letztgenannte Datei befindet sich anschließend auto-

matisch im Editor, so daß wir das Hauptprogramm bearbeiten können. Damit das Programm auch als Hauptprojektname behandelt wird, legen wir über die Befehlsfolge *Compile* → *Main File* unsere Hauptdatei fest. Das sich zunächst öffnende Fenster mit einer Suffixangabe (standardmäßig .BAS) quittieren wir mit [Return]. Aus einer Dateiliste können wir anschließend unsere Hauptdatei HAUSHALT.BAS auswählen (Bild 2.2.2). Auch wenn wir in unserer späteren Programmentwicklung die Funktionsbibliothek TOOLS.BAS im Editor bearbeiten, wird bei jedem Übersetzen (Kompilieren) die Datei HAUSHALT.BAS als Hauptdatei übersetzt.

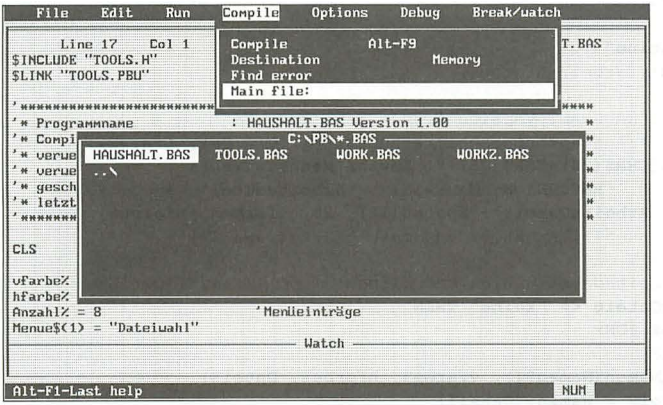

Bild 2.2.2: Das Programm HAUSHALT.BAS

Dateiwahl im Hauptprogramm

Zunächst wollen wir unser Hauptprogramm aufbereiten, indem wir den Demonstrationsaufruf der Eingaberoutine am Programmende entfernen. Außerdem wollen wir den Aufruf für die Dateianwahl in unsere Hauptschleife (*SELECT-CASE*) für die Menüauswahl integrieren. Danach ergibt sich ein Quelltext des Hauptprogramms wie in Listing 2.2.1 dargestellt. Beachten Sie, daß der geänderte Quelltext durch Fettdruck hervorgehoben ist. Auf diese Stellen sollten Sie sich besonders konzentrieren.

```
$INCLUDE "TOOLS.H"              'Deklarationen der Unit TOOLS
$LINK "TOOLS.PBU"               'Benutzerbibliothek TOOLS einbinden

'*********************************************************************
'* Programmname              : HAUSHALT.BAS Version 1.00             *
'* Compiler                  : Power Basic 2.10                      *
'* verwendete Header-Dateien: TOOLS.H                                *
'* verwendete Units          : TOOLS.PBU                             *
'* geschrieben               : April/Mai 1991                        *
'* letzte Änderung           : 16.05.1991                            *
'*********************************************************************
```

```
$STACK 4000
CLS                          'Hauptprogramm HAUSHALT

vfarbe% = 7                  'Daten initialisieren
hfarbe% = 1
Anzahl% = 8                  'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
Menue$(3) = "Datenauswertung"
Menue$(4) = "Druck"
Menue$(5) = "Grafik"
Menue$(6) = "OS-Shell"
Menue$(7) = "Hilfe"
Menue$(8) = "Ende"

ArbeitsDatei$="keine"
CALL Hintergrund(vfarbe%, hfarbe%)
WHILE Wahl% <> 8
  CALL InfoZeile("HAUSHALT Version 1.00 - Einnahmen-"+_
                 "/Ausgabenüberschußrechnung", 1, vfarbe%, hfarbe%)
  CALL InfoZeile(CHR$(24)+CHR$(25)+": Anwahl Menüeintrag"+_
                 "   RETURN: Menüauswahl    Arbeitsdatei: "+_
                 ArbeitsDatei$ + Space$(12), 25, vfarbe%, hfarbe%)
  CALL Auswahl(Menue$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
  SELECT CASE Wahl%
    CASE 1   'Arbeitsdatei wählen und öffnen
      IF ArbeitsDatei$ <> "keine" THEN
          CLOSE #DNr
        END IF
      CALL DateiNamen(Namen$(), vfarbe%, hfarbe%, Datei%)
      ArbeitsDatei$ = Namen$(Datei%)
      DNr = FREEFILE
      OPEN ArbeitsDatei$ FOR RANDOM AS #DNr LEN=50
    CASE 2   'Dateneingabe
    CASE 3   'Datenauswertung
    CASE 4   'Ergebnisse ausdrucken
    CASE 5   'Grafik
    CASE 6   'Tools (OS-Shell)
    CASE 7   'Hilfsfunktion
    CASE ELSE
  END SELECT
WEND
CLOSE #DNr
COLOR 7, 0
CLS
PRINT "Programm beendet..."
END
```

Listing 2.2.1: Hauptprogramm für zweite Sitzung

In unserem erweiterten Hauptprogramm haben wir nun eine neue Variable mit dem Namen *ArbeitsDatei$* eingeführt, der am Anfang unmittelbar die Zeichenkette *"keine"* zugewiesen wird. Anhand dieser Initialisierung erkennen wir, daß noch keine Arbeitsdatei gewählt ist. Haben wir später

eine Datei aus dem Auswahlmenü gewählt, wird in dieser Zeichenkettenvariable der Dateiname gespeichert. Eine Ausgabe in der unteren Bildschirmzeile erfolgt mit Hilfe der Routine *InfoZeile*, ebenfalls unter Verwendung dieser Variablen.

Mit Hilfe des Dateinamens ist es uns ferner möglich, zu kontrollieren, ob bereits eine Arbeitsdatei geöffnet ist. Ist nämlich der Dateiname nicht mehr *"keine"*, so wurde bereits zuvor eine aktuelle Datei gewählt. In diesem Fall muß die alte Datei, um eine Fehlermeldung zu vermeiden, zunächst geschlossen werden. Dies wird über die nachfolgend noch einmal aufgeführte IF-Abfrage realisiert.

```
IF ArbeitsDatei$ <> "keine" THEN
  CLOSE #DNr
END IF
```

Im Anschluß daran können wir die Dateiauswahl mit der benutzerdefinierten Routine *DateiNamen* durchführen. Das Unterprogramm werden wir noch gesondert besprechen. Haben wir die Datei ausgewählt, wird über die Funktion *FREEFILE* eine freie Dateinummer ermittelt und die Datei mit dieser Nummer durch eine *OPEN*-Anweisung geöffnet. Da wir in unserer Datei jeden Datensatz gezielt bearbeiten wollen, wählen wir eine Direktzugriffsdatei, deren Datensatzlänge 50 Byte betragen soll.

Das Unterprogramm zur Dateiauswahl

Wir wollen in unserem Unterprogramm *DateiNamen* eine Liste mit möglichen Arbeitsdateien verwalten und über unsere bereits in unserer ersten Sitzung formulierte Auswahlroutine anwählen. Welches Element aus unserer Liste gewählt wurde, soll an unser Hauptprogramm zurückgegeben werden. Der Unterprogrammkopf kann dann wie folgt formuliert werden:

```
SUB DateiNamen(Namen$(), vfarbe%, hfarbe%, Wahl%)
```

Namen$() ist ein Datenfeld, das die Dateiliste enthält, *vfarbe%* und *hfarbe%* sind Variablen für die Farbeinstellung der Bildschirmausgaben, die in der Routine *Auswahl* benötigt werden, und in der Variablen *Wahl%* wird die Nummer des gewählten Listenelements zurückgegeben. Die gewählte Arbeitsdatei läßt sich dann über folgende Befehlszeile zuweisen:

```
ArbeitsDatei$ = Namen$(Wahl%)
```

Den kompletten Aufbau der Routine *DateiNamen* entnehmen Sie bitte dem Listing 2.2.2. Der Quelltext der Routine wird im Anschluß an unser Hauptprogramm, getrennt durch Kommentarzeilen, in den Editor von Power Basic eingegeben.

```
'*********************************************************************
'* Unterprogramme und Funktionen für das Programm HAUSHALT           *
'*********************************************************************

SUB DateiNamen(Namen$(), vfarbe%, hfarbe%, Wahl%)
  Wahl% = 0
  Anzahl% = 0                          'Dateieinträge ermitteln
  Datei$ = DIR$("*.DAT")               'Suchmaske festlegen
  WHILE Datei$<>"" AND Anzahl% < 10    'Einträge einlesen
    Anzahl% = Anzahl%+1
    Namen$(Anzahl%) = Datei$
    Datei$ = DIR$
  WEND
  a$ = ""
  IF Anzahl% = 0 THEN                  'keine Datei gefunden
     Namen$(1) = "HAUSHALT.DAT"        'Standardname erzeugen
     Wahl%= 1
    ELSEIF Anzahl%<10 THEN
     Anzahl% = Anzahl% + 1             'neue Datei erzeugen?
     Sonder$ = "neue Datei"
     Namen$(Anzahl%) = Sonder$
  END IF
  WHILE Wahl%=0                        'Arbeitsdatei ermitteln
    CALL Hintergrund (vfarbe%, hfarbe%)
    CALL InfoZeile("Info: Bitte wählen Sie eine Arbeitsdatei aus dem"+_
                   " folgenden Menü...", 1, vfarbe%, hfarbe%)
    ARRAY SORT Namen$() FOR Anzahl%    'Dateien alphabetisch sortieren
    CALL Auswahl (Namen$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
    IF RTRIM$(Namen$(Wahl%)) = Sonder$ THEN
      CALL Fenster(10, 30, 44, 1, vfarbe%, hfarbe%)
      Namen$(Wahl%) = RIGHT$(Date$, 4)
      CALL Eingabe("Bitte neuen Dateinamen eingeben: ", Namen$(Wahl%), 8,_
                   11, 32, vfarbe%, hfarbe%)
      IF RTRIM$(Namen$(Wahl%)) = "" THEN
        Namen$(Wahl%) = "HAUSHALT"
      END IF
      Namen$(Wahl%) = Namen$(Wahl%) + ".DAT"
    END IF
  WEND
  CALL Hintergrund (vfarbe%, hfarbe%)
END SUB
```

Listing 2.2.2: Unterprogramm DateiNamen

Da die Routine sehr ausführlich im Quelltext kommentiert ist, wollen wir uns an dieser Stelle auf spezielle Erläuterungen beschränken.

Vergleichen Sie die Parameterliste bei dem Unterprogramm *DateiNamen* mit der Parameterliste beim Aufruf im Hauptprogramm. Sie werden sehen, daß die Variablentypen (z.B. Ganzzahl *Wahl%* und *Datei%*), nicht aber die Dateinamen übereinstimmen müssen. Sie sehen auch, daß innerhalb unserer benutzerdefinierten Routine wiederum bereits zuvor entwickelte Unterprogramme aufgerufen werden können. Auf diesem Wege lassen sich aus einfachen Grundfunktionen immer komplexere Routinen entwickeln.

⚠ **Versuchen Sie nicht, einen komplizierten Sachverhalt unbedingt in einem Unterprogramm zu lösen, sondern teilen Sie diesen Sachverhalt in sinnvolle Teilbereiche auf. Dadurch ist Ihr Programm einfacher zu lesen und Teilbereiche lassen sich besser austesten und überarbeiten.**

Wesentlicher Bestandteil der Routine *DateiNamen* ist die Power-Basic-Anweisung *DIR$*. Durch den ersten Aufruf *Datei$=DIR$("*.DAT")* legen wir die Suchmaske fest, d.h., alle Dateien in unserem aktuellen Verzeichnis, die über das Suffix .DAT verfügen, sollen in unserer Auswahlroutine angeboten werden. In einer *WHILE-WEND*-Schleife werden anschließend bis zu 10 Dateieinträge im Verzeichnis gesucht und unserem Datenfeld *Namen$()* zugeordnet.

Um einen Fehler in unserer Auswahlroutine zu vermeiden, initialisieren wir das erste Array-Element mit dem Namen *"HAUSHALT.DAT"*, wenn keine Datei der vorgegebenen Suchmaske entspricht (dann *Anzahl%=0*). Solange weniger als zehn Arbeitsdateien im Verzeichnis gefunden werden, wird die Möglichkeit geschaffen, neue Dateien anzulegen (dann *Anzahl%<10*). Dazu wird ein weiteres Element unseres Datenfelds mit dem Eintrag "*neue Datei*" initialisiert. Wird dieser Eintrag später durch die Auswahlroutine gewählt, wird eine Eingaberoutine für Dateinamen aufgerufen (dann *RTRIM$(Namen$(Wahl%))=Sonder$*). Die Funktion *RTRIM$* entfernt die rechts vom Dateinamen vorhandenen Leerzeichen, die von der Routine *Auswahl* angehängt werden, damit sämtliche Listeneinträge eine einheitliche Zeichenkettenlänge aufweisen.

Bevor wir die Auswahlroutine aufrufen, wird zunächst unser Datenfeld über die ARRAY-Anweisung alphabetisch sortiert. Beachten Sie, daß wir lediglich die bereits initialisierten Einträge (*ARRAY SORT Namen$(), FOR Anzahl%)* sortieren lassen.

Bild 2.2.3: Dateiauswahl im Programm HAUSHALT

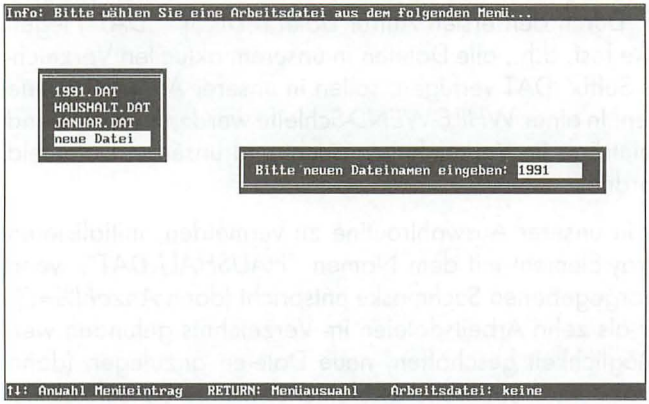

Bild 2.2.4: Eingabe eines neuen Dateinamens bei der Dateiauswahl

Wie sich unsere erstellte Routine *DateiNamen* auf dem Bildschirm darstellt, wird in den Bildern 2.2.3 und 2.2.4 deutlich.

Workshop

Checkliste

1. Welche DOS-Befehle sind in Power Basic realisiert?
2. Welche Dateitypen haben Sie kennengelernt und wo liegen die wesentlichen Anwendungsbereiche?
3. Erläutern Sie die Regeln zur Dateinamensvergabe unter Power Basic!
4. Welche Aufgaben sind in einer Listenverwaltung durchzuführen und welche Befehle stehen hierzu in Power Basic zur Verfügung?

Ideen für eigene Übungen

- Erweitern Sie die Routine zur Dateiwahl so, daß Sie auch das Verzeichnis wechseln können.
- Erweitern Sie die Routine so, daß Sie auch Informationen über die Dateigröße bei der Auswahl erhalten (ähnlich dem DOS-DIR-Befehl).

2.3 Dritte Sitzung: Dateneingabe der Einnahmen und Ausgaben

Die Themen dieser Sitzung:

- Bildschirmausgaben
- Programmieren einer Eingabemaske
- Verknüpfung der Eingabe mit der Datenverwaltung
- Felddefinition von Datensätzen
- Abfangen von Laufzeitfehlern

Information

In dieser Sitzung wollen wir die für unsere Einnahmen-/Ausgabenverwaltung benötigten Daten über eine einfache Eingaberoutine einlesen lassen. Die Eingaben sollen nunmehr kontrolliert erfolgen, d.h., an den Stellen, an denen wir Zahlen eingeben müssen, sollen keine Buchstaben eingebbar sein. Zu diesem Zweck werden wir unsere in der ersten Sitzung erstellte Eingaberoutine überarbeiten und dementsprechend erweitern. Da wir zu jedem Datensatz, d.h. zu jeder Buchung, mehrere Informationen benötigen, werden wir eine Eingabemaske konstruieren, die es uns ermöglicht, zwischen den unterschiedlichen Eingabefeldern zu wechseln. Dazu ist es bereits erforderlich, die Datenstruktur unserer Arbeitsdatei genauer festzulegen. Wir müssen uns entscheiden, welche Information wir zu jeder Buchung speichern wollen.

Um Fehler während der Laufzeit des Programms zu vermeiden, müssen wir ferner eine Möglichkeit schaffen, die Dateneingabe nicht durchführen zu lassen, solange noch keine Arbeitsdatei gewählt wurde.

Da wir uns für eine Direktzugriffsdatei entschieden haben, in der unsere Daten gespeichert werden sollen, ist es sinnvoll, die Dateneingabe mit der Datenverwaltung in unserer Arbeitsdatei zu koppeln. Aus diesem Grund werden wir in dieser Sitzung bereits das Grundgerüst für die Dateiverwaltung integrieren, das Schwerpunktthema unserer nächsten Sitzung sein wird.

Bildschirmausgaben

In Power Basic lassen sich über die Befehle *COLOR, LOCATE* und *PRINT* sehr einfach ansprechende Bildschirmausgaben vornehmen. So können wir über *COLOR* die Farbe der Ausgaben festlegen, mit *LOCATE* können wir den Textcursor an eine bestimmte Stelle des Bildschirmes positionieren und mit *PRINT* lassen sich Informationen ausgeben. Von diesen Möglichkeiten haben wir bereits in unserer Funktionsbibliothek ausgiebigen Gebrauch gemacht.

Erstellung einer Eingabemaske

Im Vergleich zur einfachen Eingabe, wie wir sie z.B. bei der Neuanlage von Arbeitsdateien kennengelernt haben, können in einer Eingabemaske parallel mehrere Eingaben gemacht werden. Zwischen den Eingabefeldern, die in einer Maske definiert werden, kann über Tastaturbefehle gewechselt werden, so daß einzelne Eingaben mehrmals editierbar sind. Die einzelnen Eingabefelder gehören dabei zusammen, wie z.B. die Datenfelder eines Datensatzes. Die einfachste Möglichkeit, eine Maske zu erstellen, ist die Kombination mehrerer *INPUT*-Anweisungen. Vor jedem *INPUT* wird der Cursor dabei an die entsprechende Bildschirmposition plaziert. Die Unzulänglichkeiten dieser Power-Basic-Anweisung haben wir allerdings schon kennengelernt (keine Eingabefeldbegrenzung). Da wir bereits über eine einfache Eingaberoutine verfügen, werden wir die *INPUT*-Anweisung wieder durch unsere benutzerdefinierte Eingaberoutine *Eingabe* ersetzen.

Verknüpfung der Eingabe mit der Datenverwaltung

Bei der Wahl von Direktzugriffsdateien ist eine unmittelbare Verknüpfung mit der Datenverwaltung sinnvoll. Jeden Datensatz können Sie unmittelbar nach der Bearbeitung gezielt in die Arbeitsdatei schreiben. Ihre Datei ist damit immer auf dem aktuellen Stand, selbst wenn Ihr Rechner durch einen Stromausfall oder einen Systemabsturz keine Arbeiten mehr durchführen kann. Würden Sie z.B. mit einer sequentiellen Datei arbeiten, müßte der komplette Datenbestand in den Speicher eingelesen werden. Nach der Bearbeitung der gesamten Datei könnte dann der Datenbestand wieder in die sequentielle Datei geschrieben werden. Durch dieses Vorgehen ersparen Sie sich die zeitaufwendigen Lese- und Schreibvorgänge. Fällt Ihr Rechner allerdings während der Bearbeitungsphase aus, gehen sämtliche gemachten Dateiänderungen unwiderruflich verloren.

An dieser Stelle soll noch ein weiterer Nachteil der sequentiellen Dateien angeführt werden. Wird der Gesamtinhalt im Speicher verarbeitet, so ist die Dateigröße, anders als bei Randomdateien, auf die Arbeitsspeichergröße beschränkt. Eine unmittelbare Verknüpfung der Eingaberoutine und der Datenverwaltung ist bei sequentiellen Dateien nicht sinnvoll.

Felddefinition von Datensätzen

In jedem Datensatz können mehrere Informationen gespeichert werden. Somit lassen sich die durch eine Eingabemaske erhaltenen, zusammengehörigen Ergebnisse unmittelbar in einen gemeinsamen Block (Datensatz) schreiben und später zusammenhängend wieder lesen. Die Aufteilung in sogenannte Datenfelder erfolgt dabei durch die *FIELD*-Anweisung. Wollen Sie z.B. eine Namensliste in einer Randomdatei speichern, wobei die Vornamen und Nachnamen in getrennten Feldern verwaltet werden, müssen Sie sich zunächst Klarheit über die benötigten Feldlängen verschaffen. Gehen wir davon aus, daß der derzeit längste Vorname aus 30 Zeichen und der längste Nachname aus 40 Zeichen besteht. Zur Reserve addieren Sie für jedes Feld 5 Zeichen hinzu, so daß sich folgende Anforderungen ergeben:

```
Vorname     : max. 35 Zeichen
Nachname    : max. 45 Zeichen
Gesamtlänge:       80 Zeichen
```

Um die Datei zu öffnen und die Felder dementsprechend zu definieren, könnten Sie folgenden Quelltext einsetzen:

```
'freie Dateinummer ermitteln
Nr = FREEFILE
'Randomdatei öffnen und Datensatzlänge
'auf 80 Zeichen festlegen
OPEN "TEST.DAT" FOR RANDOM AS #Nr LEN=80
'Datenfelder definieren
FIELD #Nr, 35 AS VorName$, 45 AS NachName$
```

Abfangen von Laufzeitfehlern

Bei der Programmierung sollte man Probleme, die sich ergeben können, unmittelbar berücksichtigen. Zu diesem Zweck bietet sich die Einführung von Steuerungsvariablen an. Abhängig von diesen Variablen kann dann die Programmausführung gesteuert werden. Dabei spielen vor allen Dingen Laufzeitfehler eine Rolle, Fehler also, die erst in Sonderfällen bei der Programmausführung auftreten und nicht unmittelbar bei der Programmentwicklung sichtbar werden. Bedenken Sie, daß die an eigene

Routinen übergebenen Werte nicht immer kontrolliert werden. In unserem Fall würde die Dateneingabe, falls noch gar keine Arbeitsdatei ausgewählt und geöffnet wurde, zu einem Laufzeitfehler führen. In unserem Programm HAUSHALT werden wir daher die Dateneingabe nur zulassen, wenn zuvor eine Arbeitsdatei aktiviert wurde.

Praxis

Haben wir uns in den ersten Sitzungen ausführlich mit dem Hauptprogramm befaßt, wollen wir an dieser Stelle lediglich die Programmzeilen anführen, die in der *SELECT-CASE*-Anweisung zu ergänzen sind.

```
:'unveränderter Quelltext
:
:
CASE 2   'Dateneingabe
  IF ArbeitsDatei$="keine" THEN
     CALL KeineDateiAktiv(vfarbe%, hfarbe%)
    ELSE
     CALL DatenEingabe(DNr, vfarbe%, hfarbe%)
  END IF
CASE 3   'Datenauswertung
:
:
:unveränderter Quelltext
```

Listing 2.3.1: Erweiterung des Hauptprogramms HAUSHALT.BAS für die Dateneingabe

Über die Steuerungsvariable *ArbeitsDatei$* können wir die Zulässigkeit von Eingaben überprüfen. Wurde keine Arbeitsdatei gewählt, wird durch das Unterprogramm *KeineDateiAktiv* eine Meldung für den Programmanwender ausgegeben, ansonsten wird ein Unterprogramm für die Dateneingabe aufgerufen. Die Unterprogramme werden wir später noch genauer besprechen. Auf diesem Wege vermeiden wir das Auftreten von Laufzeitfehlern.

Die Erweiterung der Eingaberoutine

Wie bereits erwähnt, wollen wir in dieser Sitzung unsere Eingaberoutine optimieren. Die veränderte Routine ist in Listing 2.3.2 abgedruckt, gemachte Änderungen sind durch Fettdruck hervorgehoben. Innerhalb der Benutzeroberfläche von Power Basic müssen Sie, um die Routine bearbeiten zu können, die Datei TOOLS.BAS laden (Bild 2.3.1).

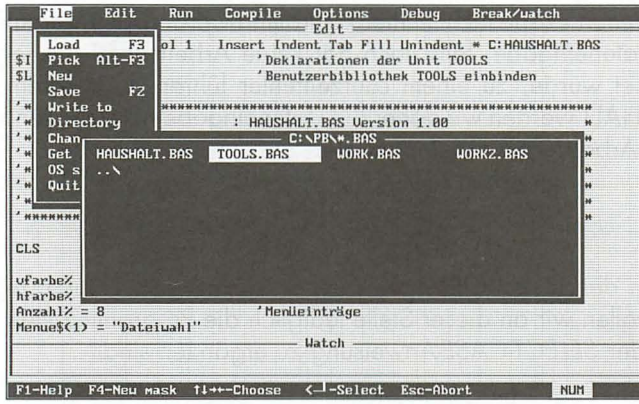

Bild 2.3.1: Laden der Datei TOOLS.BAS

Power Basic trägt Dateien in sogenannte Pick-Listen ein, d.h., Sie können über die Tastenkombination [Alt] + [F3] bzw. über die Menüfolge *File* → *Pick* eine bestimmte Quelldatei, die zu einem Hauptprogramm gehört, in den Editor laden. Da der Editor von Power Basic nur Quelltexte mit 64 Kbyte Umfang, der Compiler aber weitaus größere Dateien verarbeiten kann, haben Sie damit die Möglichkeit, auch mit gesplitteten Programmen komfortabel zu arbeiten. Dazu sollten Sie, wie bereits in der zweiten Sitzung beschrieben, den *Main file* mit dem Namen Ihres Hauptprogramms, in diesem Fall HAUSHALT.BAS, festgelegt haben.

Über *Pick* können Sie recht schnell zwischen einzelnen Programm-Modulen springen, in unserem Fall zwischen dem Hauptprogramm und der Funktionsbibliothek. Bei einem *Kompilier*-Befehl wird nun immer das Programm HAUSHALT.BAS übersetzt.

⚠ **Beachten Sie, daß wir in unserem Hauptprogramm immer die fertig kompilierte Unit mit dem Kompilier Befehl *$LINK* "TOOLS.PBU" einbinden. Nehmen wir Änderungen an dieser Unit vor, müssen wir sie auch getrennt neu übersetzen. Alternativ könnte man in einem Hauptprogramm den Quelltext der Bibliothek mit *$INCLUDE* einbinden, dann würde der Quelltext allerdings bei jedem Übersetzungsvorgang erneut übersetzt, auch dann, wenn gar keine Änderungen vorgenommen wurden.**

Der Umgang mit den Pick-Listen ist zunächst ein wenig gewöhnungsbedürftig, aber bei einiger Übung werden Sie diese verstärkt einsetzen. Die Anwendung dieser Möglichkeit ist wahlfrei, so daß Sie, wenn Sie mit dieser Handhabung nicht zurechtkommen, jede Quelldatei eigenständig im Editor behandeln können.

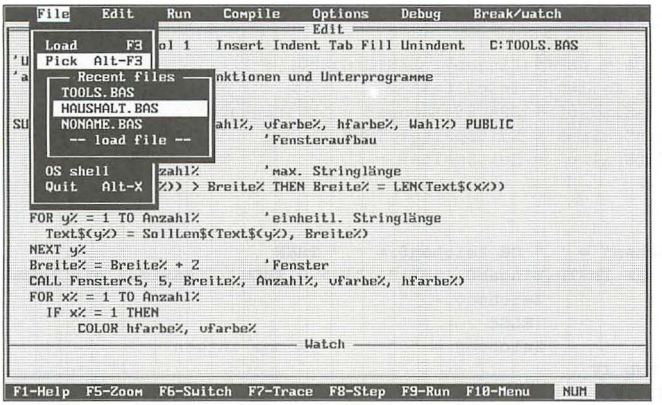

Bild 2.3.2: Rückkehr zu HAUSHALT.BAS über die Pick-Liste

Nun wollen wir aber die wesentlichen Änderungen der Eingaberoutine besprechen. Bei einer Erweiterung von benutzerdefinierten Routinen sollten Sie darauf achten, möglichst keine Änderungen an den Parameterlisten vorzunehmen. Dann kann die Funktion wie bisher angewendet werden und ältere Programme können mit der erweiterten Routine, unmittelbar und ohne weitere Änderungen der Programmaufrufe im Hauptprogamm vorzunehmen, eingesetzt werden.

```
SUB Eingabe (Vorgabe$, Eingab$, Laenge%, Zeile%, Spalte%, vfarbe%,
hfarbe%) PUBLIC
   IF LEFT$(Vorgabe$, 2) = "I|" THEN        'Beschränkung auf Ganzzahlen
        Zulaessig$ = "-+1234567890"
        Vorgabe$ = RIGHT$(Vorgabe$, LEN(Vorgabe$) - 2)
      ELSEIF LEFT$(Vorgabe$, 2) = "R|" THEN  'Fließkommazahl
        Zulaessig$ = "-+.,1234567890"
        Vorgabe$ = RIGHT$(Vorgabe$, LEN(Vorgabe$) - 2)
    END IF
  LOCATE Zeile%, Spalte%
  COLOR vfarbe%, hfarbe%
  PRINT Vorgabe$;
  COLOR hfarbe%, vfarbe%
  PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
  MinSpalte% = LEN(Vorgabe$) + Spalte%
  MaxSpalte% = LEN(Vorgabe$) + Spalte% + Laenge% - 1
  EditSpalte% = LEN(Vorgabe$) + Spalte% + LEN(Eingab$)
  IF EditSpalte%>MaxSpalte% THEN EditSpalte%= EditSpalte% - 1
  WHILE Zeichen$ <> CHR$(13)
    LOCATE Zeile%, EditSpalte%, 1
    Zeichen$ = INKEY$
    IF LEN(Zeichen$) = 1 THEN
      SELECT CASE ASC(Zeichen$)
        CASE 8   'Backspace
          IF EditSpalte% > MinSpalte% THEN
            IF LEN(Eingab$) = Laenge% THEN
```

```
          EditSpalte% = EditSpalte% + 1
        END IF
        Eingab$ = LEFT$(Eingab$, LEN(Eingab$) - 1)
        LOCATE Zeile%, MinSpalte%
        PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
        IF MinSpalte% < EditSpalte% THEN
          EditSpalte% = EditSpalte% - 1
        END IF
      END IF
    CASE 32 TO 254
      IF Zeichen$ = "." OR Zeichen$ = "," THEN
            IF INSTR(Eingab$, ANY ".,") > 0 THEN   'maximal ein Komma
                    Zeichen$ = ""
                     ELSE
                    Zeichen$ = "."        'Komma intern
          END IF
        END IF
        IF Zulaessig$ <> "" THEN                  'Zeichen zulässig?
          IF INSTR(Zulaessig$, Zeichen$) = 0 THEN
            Zeichen$ = ""                         'nein!
           ELSEIF EditSpalte% > MinSpalte% THEN 'evtl. nicht in
                                                  'der Position
            IF INSTR("+-", Zeichen$) > 0 THEN
              Zeichen$ = ""                       'Vorzeichen nur an
            END IF                                ' erster Position
          END IF
        END IF
        IF Zeichen$<>"" THEN
      IF EditSpalte% <= MaxSpalte% THEN
        IF LEN(Eingab$) = Laenge% THEN
          Eingab$ = LEFT$(Eingab$, LEN(Eingab$) - 1)
        END IF
        Eingab$ = Eingab$ + Zeichen$
        LOCATE Zeile%, MinSpalte%
        PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
        IF MaxSpalte% > EditSpalte% THEN
          EditSpalte% = EditSpalte% + 1
        END IF
      END IF
    END IF
  CASE ELSE
 END SELECT
END IF
IF Zeichen$ <> CHR$(13) THEN Zeichen$ = ""
WEND
END SUB
```

Listing 2.3.2: Die überarbeitete Eingaberoutine

In dem am Beginn des Unterprogramms eingefügten *IF*-Block werden die zulässigen Eingabezeichen behandelt. Um dem Hauptprogramm die Möglichkeit zu geben, ohne die Einführung eines neuen Unterprogramm-

parameters, Eingabezeichen zu definieren, wird die Variable *Vorgabe$* in ihrer Anwendung erweitert. Wird die Zeichenkette, die an *Vorgabe$* übergeben wird, durch *"I I"* oder *"R I"* eingeleitet, sind nur Zahlen, d.h. entweder Ganzzahlen oder Fließkommazahlen, ansonsten sämtliche alphanumerischen Zeichen bei der Eingabe erlaubt. Dieses Definitionsmerkmal wird, falls es vorhanden ist, durch die Anweisung *Vorgabe$ = Right$(Vorgabe$,LEN(Vorgabe$)-2)* direkt wieder entfernt.

Um die Information weiter zu nutzen, wird die lokale Variable *Zulaessig$* eingeführt, in der die zulässigen Eingabezeichen definiert werden. Gegenüber den Ganzzahlen (*"I I"*: Typ Integer) wird bei den Fließkommazahlen (*"R I"*: Typ Real) zusätzlich der Dezimalpunkt zugelassen. Außerdem soll die Möglichkeit gegeben werden, sowohl positive als auch negative Zahlen einzugeben.

Am Ende des Unterprogramms müssen nun noch der Dezimalpunkt und die Vorzeichen gesondert behandelt werden. Da wir sämtliche Eingaben in Zeichenketten vornehmen, müssen wir numerische Werte später in Zahlenwerte konvertieren. Da Power Basic über diese Konvertierroutinen bereits verfügt, ist es sinnvoll, die Zeichenkette nur mit korrekter Formatierung, die anschließend eine fehlerfreie Umsetzung erlaubt, eingeben zu lassen. Dazu wird das jeweilig über die *INKEY$*-Anweisung eingelesene Zeichen, bevor eine Zeichenkette für die Eingabe aufgebaut wird, auf Zulässigkeit überprüft. Da in Deutschland das Dezimalkomma gebräuchlich ist, Power Basic intern aber mit dem amerikanischen Dezimalpunkt arbeitet, wird ein wahlfrei eingegebenes Komma automatisch durch die nachfolgenden Zeilen in einen Dezimalpunkt umgesetzt. Zusätzlich wird durch die *INSTR*-Funktion dafür gesorgt, daß höchstens ein Dezimalpunkt innerhalb einer Eingabe vorhanden ist.

```
IF Zeichen$ = "." OR Zeichen$ = "," THEN
  IF INSTR(Eingab$, ANY ".,") > 0 THEN
    Zeichen$ = ""
   ELSE
    Zeichen$ = "."
  END IF
END IF
```

Im Anschluß daran wird die Überprüfung auf zulässige Zeichen vorgenommen. Ist das Zeichen nicht erlaubt, also nicht innerhalb von *Zulaessig$* enthalten, so wird es gelöscht und für die Eingabe nicht verwendet. Außerdem wird bei den Vorzeichen darauf geachtet, daß diese nur als

erste Zeichen in einem Eingabefeld zulässig sind (dann: *EditSpalte%=MinSpalte%*).

```
IF Zulaessig$ <> "" THEN
  IF INSTR(Zulaessig$, Zeichen$) = 0 THEN
    Zeichen$ = ""
  ELSEIF EditSpalte% > MinSpalte% THEN
    IF INSTR("+-", Zeichen$) > 0 THEN
      Zeichen$ = ""
    END IF
  END IF
END IF
```

⚠ **Beachten Sie, daß in der derzeitigen Unterprogrammversion die Beschränkungen des Kommas und der Vorzeichen auch für alphanumerische Eingaben gelten. Eine weitere Optimierung der Routine ist also durchaus möglich.**

Wenn Sie die Datei nun im Editor auf den neuesten Stand gebracht und in eine Unit kompiliert haben (dazu sollten Sie den *Main File* zunächst entfernen), können Sie über die Pick-Liste zu dem Programm HAUSHALT.BAS, wie oben beschrieben, zurückkehren (Bild 2.3.2).

Unterprogramme für die Dateneingabe

Wie bereits im Hauptprogramm eingefügt, benötigen wir die Unterprogramme *KeineDateiAktiv* und *Dateneingabe*. Diese geben wir in den Editor unmittelbar nach dem in unserer zweiten Sitzung erstellten Unterprogramm *DateiNamen* ein. Für die Dateneingabe benötigen wir das deutsche Datum (wie wir später noch sehen werden) und ein Unterprogramm, um die Bildschirmmaske zu aktualisieren. Da diese Routinen (Listing 2.3.2) recht einfach zu verstehen sind, wollen wir sie hier kurz besprechen.

In der Routine *KeineDateiAktiv* wird die Meldung auf dem Bildschirm ausgegeben, daß zunächst eine Arbeitsdatei gewählt werden muß. Diese Meldung bleibt so lange in einem Bildschirmfenster (*CALL Fenster*) stehen, bis eine beliebige Taste gedrückt wird (*CALL Taste*).

```
SUB KeineDateiAktiv(vfarbe%,hfarbe%)
  COLOR hfarbe%, vfarbe%
  CALL Fenster(10, 30, 48, 1, vfarbe%, hfarbe%)
  LOCATE 11, 32, 0
  PRINT "Bitte wählen Sie zunächst eine Arbeitsdatei..."
  CALL Warte
  COLOR vfarbe%, hfarbe%
END SUB
SUB AktualisiereMaske(SatzNummer&, Datum$, Kommentar$, Wert$,_
                      vfarbe%, hfarbe%)
```

```
  COLOR vfarbe%, hfarbe%
  LOCATE  9, 31, 0 : PRINT SatzNummer&; SPACE$(10)
  LOCATE 11, 31 : PRINT Datum$
  LOCATE 13, 31 : PRINT SollLen$(Kommentar$, 36)
  LOCATE 15, 31 : PRINT SollLen$(Wert$,15)
END SUB
```

Listing 2.3.3: Allgemeine Unterprogramme für die Dateneingabe

Unser Datensatz soll aus den Feldern Buchungsdatum, Kommentar zur Buchung und der Buchungssumme bestehen. Die Feldinhalte müssen jeweils in unsere Bildschirmmaske eingetragen werden. Die Maske selbst ist ganz gezielt über Testläufe angepaßt worden. Es wäre also sinnvoll, den Aufbau einer Bildschirmmaske über Unterprogramme weiter zu vereinfachen. Die Routine *AktualisiereMaske* schreibt die später aus der Datei gelesenen Informationen gezielt in unsere markierten Eingabefelder, wie sie bei der Dateneingabe definiert wurden. Außerdem wird die jeweils aktuelle Datensatznummer, die intern zu ermitteln ist, angezeigt.

Die Dateneingabe

Um die Programmierung zu vereinfachen, sollten Sie, auch wenn eine Verknüpfung der Dateneingabe und der Verwaltung dieser Daten in Direktzugriffsdateien sinnvoll wäre, keine parallele Entwicklung dieser Routine vornehmen. Wir werden uns zunächst lediglich auf die Dateneingabe beschränken (Listing 2.3.3) und die Routine, ähnlich wie bereits unsere allgemeine Eingaberoutine, in der nächsten Sitzung um spezielle Anweisungen erweitern.

```
SUB DatenEingabe(DateiNummer, vfarbe%, hfarbe%)
  FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
        4 AS Betrag$
  CALL Hintergrund(vfarbe%, hfarbe%)
  CALL Fenster(8, 10, 60, 8, vfarbe%, hfarbe%)
  CALL InfoZeile("PgUp/PgDn: Datensätzen blättern   Return: Buchung"+_
                 " vornehmen  ESC: Eingabeende", 25, vfarbe%, hfarbe%)
  LOCATE 9, 12   'Eingabemaske ausgeben
  PRINT "Datensatznummer : ";
  LOCATE 11, 12
  PRINT "Buchungsdatum   : ";
  LOCATE 13, 12
  PRINT "Beschreibung    : ["+SPACE$(36)+"]";
  LOCATE 15, 12
  PRINT "Einnahme/Ausgabe: ["+SPACE$(15)+"]";
  CALL Fenster(19, 10, 60, 3, vfarbe%, hfarbe%)
  LOCATE 20, 12
  PRINT "Eine Einahme wird mit einem positivem [+] und eine Ausgabe"
  LOCATE 21, 12
  PRINT "mit einem negativen Vorzeichen [-] in der ersten Spalte"
```

```
  LOCATE 22, 12, 0
  PRINT "gekennzeichnet. Die Verwendung des [+] ist dabei wahlfrei."
  'hier muß u.U. ein zu Beginn eingelesener
  'Datensatz initialisiert und an die
  'Eingabemaske übergeben werden
  EditFeld% = 0
  WHILE a$<>CHR$(27)
    a$ = INKEY$                          'ESC Eingabeende
    b$=RIGHT$(a$, 1)
    IF b$=CHR$(81) THEN                  'PgUp
        'hier wird später in der Datei
        'zurückgeblättert, die Daten werden
        'aus der Datei eingelesen und zuge-
        'wiesen
      ELSEIF b$=CHR$(73) THEN            'PgDn
        'hier wird später weitergeblättert,
        'Daten werden aus der Datei gelesen
        'und zugewiesen
      ELSEIF a$=CHR$(13) THEN            'Return
        EditFeld% = EditFeld% XOR 1      'editiere 1. bzw. 2.Feld
        IF EditFeld%=1 THEN
            Kommentar$ = RTRIM$(Kommentar$)
            CALL Eingabe("", Kommentar$, 36, 13, 31, vfarbe%, hfarbe%)
          ELSE
            Ein$ = RTRIM$(Ein$)
            CALL Eingabe("R|", Ein$, 15, 15, 31, vfarbe%, hfarbe%)
        END IF
    END IF
    IF b$=CHR$(73) OR b$=CHR$(81) OR a$=CHR$(13) THEN
      CALL AktualisiereMaske (SatzNummer&, Datum$, Kommentar$,_
                              Ein$, vfarbe%, hfarbe%)
    END IF
  WEND
  CALL HinterGrund(vfarbe%, hfarbe%)
END SUB
```

Listing 2.3.4: Grundgerüst der Dateneingabe

☞ Beachten Sie, daß für die Felddefinition und die Schreib- und Lesebefehle der Arbeitsdatei die Dateinummer notwendig ist, die im Hauptprogramm beim *OPEN*-Befehl vergeben wird. Aus diesem Grund muß die Dateinummer auch im Unterprogramm bekannt sein. Dies haben wir an dieser Stelle durch den speziellen Parameter *DateiNummer* im Unterprogrammkopf realisiert.

Zunächst wird in unserer Routine *DatenEingabe* der Feldaufbau für unsere Datei definiert. An diesen Feldaufbau werden wir uns bereits bei der Erstellung der Eingabemaske halten müssen. Die einzelnen Längen sind in der nachfolgenden Liste noch einmal zusammengestellt.

Datenfeldbezeichnung	Datenfeldlänge
BuchungsDatum$	10 Zeichen
ZeichenBeschreibung$	36 Zeichen
ZeichenBetrag$	4 Zeichen

Tabelle 2.3.1: Dateiformat für das Programm HAUSHALT

Unsere Datensatzlänge beträgt also genau 50 Zeichen, wie bereits im Hauptprogramm in der *OPEN*-Anweisung definiert, so daß hier keine Änderungen erforderlich sind. Das Buchungsdatum ist das gegenwärtige Systemdatum, so daß wir es automatisch eintragen lassen können. Wie, das werden wir in der nächsten Sitzung sehen. An dieser Stelle können wir uns also auf die Eingabe der Buchungsbeschreibung und des Buchungsbetrages beschränken.

☞ Das Datenfeld für die Buchungssumme besteht zwar nur aus vier Zeichen, d.h. aber nicht, daß nur vier Zahlen in das Eingabefeld eingetragen werden dürfen. Der Wert einer einfachen Fließkommazahl kann in 4 Byte verschlüsselt werden, so daß eine größere Felddimensionierung die Datei unnötig vergrößern würde. Einzelheiten hierzu werden Sie in unserer vierten Sitzung noch kennenlernen.

Zu Beginn der Routine wird die Bildschirmmaske mit zugehörigen Kurzinformationen auf den Bildschirm gezeichnet. Anschließend können wir uns direkt mit der Eingabe befassen, die innerhalb einer *WHILE-WEND*-Schleife erfolgt. Wir wollen später zwischen den Datensätzen blättern können, wozu wir die Tastaturcodes zum Blättern bereits berücksichtigen, aber noch nicht weiter behandeln. Der Abbruch der Eingabe und damit auch der Schleife, soll über die [Esc]-Taste erfolgen. Zum Umschalten des aktiven Eingabefeldes soll die [Return]-Taste verwendet werden.

Wie Sie im Listing sehen, haben wir eine Steuerungsvariable *EditFeld%* vor der Eingabeschleife eingeführt und mit dem Wert 0 initialisiert. Bei jeder *XOR*-Verknüpfung wird aus einer 0 eine 1 bzw. aus einer 1 eine 0. Das bedeutet in unserem Fall, daß der Wert von *EditFeld%* abwechselnd 0 bzw. 1 beträgt. Damit ist ein einfaches Umschalten zwischen den Eingabefeldern möglich.

```
HAUSHALT Version 1.00 - Einnahmen-/Ausgabenüberschußrechnung

Datensatznummer :   1

Buchungsdatum   :  11.05.1991

Beschreibung    : [Kraftfahrzeugsteuer                    ]

Einnahme/Ausgabe: [ 300.45        ]

Eine Einahme wird mit einem positivem [+] und eine Ausgabe
mit einem negativen Vorzeichen [-] in der ersten Spalte
gekennzeichnet. Die Verwendung des [+] ist dabei wahlfrei.

PgUp/PgDn: Datensätzen blättern   Return: Buchung vornehmen  ESC: Eingabeende
```

Bild 2.3.3: Das Eingabefenster im Programm HAUSHALT

Wie Sie sehen, sind vier Tasten in unserer Eingaberoutine von Bedeutung. Die [Esc]-, die [Return]-, die [PgUp]- und die [PgDn]-Taste. Mit Ausnahme der [Escape]-Taste sorgen die anderen Tasten für eine Änderung der Bildschirmmaske. Die Aktualisierung wird daher nur beim Betätigen einer dieser Tasten durch die Routine *AktualisiereMaske* vorgenommen. Das Aussehen der definierten Eingabemaske ist in Bild 2.3.3 dargestellt.

Workshop

Checkliste

1. Welche Anweisungen werden für ansprechende Bildschirmausgaben benötigt?
2. Wie läßt sich auf einfachem Wege eine Eingabemaske realisieren?
3. Warum sollte die Eingabe bei Randomdateien mit der Dateiverwaltung unmittelbar verknüpft werden?
4. Auf welchem Wege können Laufzeitfehler abgefangen werden?

Ideen für eigene Übungen

- Erweitern Sie die Eingaberoutine so, daß bei alphanumerischen Eingaben beliebig viele Kommata und Vorzeichen eingebbar sind!
- Erweitern Sie die Eingaberoutine so, daß für alphanumerische Zeichen ebenfalls nur ganz bestimmte Zeichen zulässig sind!

2.4 Vierte Sitzung: Datenverwaltung mit Direktzugriffsdateien

Die Themen dieser Sitzung:

- Verschlüsselung numerischer Datentypen
- Auslesen von Systemdatum und Systemzeit
- Dateiverwaltung mit Direktzugriffsdateien
- Die LOF-Funktion
- Konvertierroutinen

Information

Haben wir in den ersten drei Sitzungen bereits einen Großteil unseres Programms abgehandelt, können wir uns in dieser Phase der Programmierung gezielt mit einigen speziellen Themen der Dateiverwaltung befassen. Wir wollen unsere Eingaberoutine dahingehend erweitern, daß die Dateiverwaltung parallel zur Dateneingabe verläuft. Dazu müssen wir uns zunächst ein wenig genauer mit den Direktzugriffs- bzw. Random-Dateien befassen.

Verschlüsselung numerischer Datentypen

Die Datensätze einer Direktzugriffsdatei werden als Zeichenkette verwaltet. Dies wird auch an der *FIELD*-Anweisung, die wir in unserer letzten Sitzung bereits kennengelernt haben, deutlich.

```
FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
      4 AS Betrag$
```

Sämtliche Datenfelder werden als Zeichenketten, erkennbar durch das Datentypkennzeichen $, definiert. Das heißt allerdings nicht, daß numerische Daten in Zeichenketten, wie z.B. *"123"* abgespeichert werden müssen. Numerische Daten werden rechnerintern in ein Binärformat verschlüsselt, das nicht direkt lesbar ist, aber in Zeichenkettenform gespeichert werden kann. Dazu sind Konvertier- bzw. Umsetzungsroutinen erforderlich, die wir später noch besprechen werden. Wenn man diese

Möglichkeit nutzt, fällt die Datensatzlänge geringer aus, als es bei Zahlen im Stringformat der Fall wäre. Wie viele Bytes bestimmte Datentypen für ihre Verschlüsselung benötigen, wird in Tabelle 2.4.1 aufgelistet.

Ganzzahlen	Integer	2 Byte, z.B. a%
	Long-Integer	4 Byte, z.B. a&
	Quad-Integer	8 Byte, z.B. a&&
Fließkomma	einfachgenau	4 Byte, z.B. a!
	doppeltgenau	8 Byte, z.B. a#
	erweitert genau	10 Byte, z.B. a##
BCD-Zahlen	Festzahl	8 Byte, z.B. a@
	Fließkommazahl	10 Byte, z.B. a@@

Tabelle 2.4.1: Notwendige Anzahl von Bytes für Datentypen von Power Basic

Da wir für den Buchungsbetrag den Datentyp einer einfachen Fließkommazahl gewählt haben, benötigen wir für die Zeichenkette *Betrag$*, die diesen Wert aufnehmen soll, insgesamt 4 Byte (*4 AS Betrag$*) in unserem Datensatz.

Konvertierfunktionen

In Power Basic sind drei wesentliche Gruppen von Konvertierroutinen enthalten. Die erste Gruppe erlaubt das Kopieren von numerischen Werten in bestimmte Variablentypen, die zweite und dritte Gruppe kommt in Verbindung mit den Direktzugriffsdateien zum Einsatz. Dabei erlaubt die zweite Gruppe das Umsetzen von Stringdaten aus Randomdateien in numerische Werte und die dritte Gruppe – dem entgegengesetzt – die Umwandlung von numerischen Werten aus einer Randomdatei in Zeichenketten. Die Funktionen sind in Tabelle 2.4.1 zusammengefaßt.

Gruppe 1	Gruppe 2	Gruppe 3	Datentyp
CINT	CVI	MKI$	Integer
CLNG	CVL	MKL$	Long-Integer
CQUD	CVQ	MKQ$	Quad-Integer
CSNG	CVS	MKS$	einfachgenau
CDBL	CVD	MKD$	doppeltgenau
CEXT	CVE	MKE$	erweitert genau
CFIX	CVF	MKF$	BCD-Festzahl
CBCD	CVB	MKB$	BCD-Fließkommazahl

Tabelle 2.4.2: Hauptgruppen von Konvertierfunktionen in Power Basic

Auf die erste Gruppe kann in der Regel verzichtet werden, da Power Basic, wenn möglich, Typkonvertierungen automatisch durchführt. Für unser Programm HAUSHALT benötigen wir die zweite, um Daten in den sogenannten Dateipuffer zu kopieren, und die dritte Gruppe, um Daten aus dem Dateipuffer zu lesen.

☞ Mit der *ASC()*-Funktion kann das Zeichen eines beliebigen ASCII-Codes und mit der *CHR$()*-Funktion der ASCII-Code eines beliebigen Zeichens ermittelt werden (vgl. ASCII-Tabelle im Anhang). Mit den Funktionen *VAL* und *STR$()* können Sie Zeichenketten in numerische Äquivalente konvertieren und umgekehrt.

Dateiverwaltung mit Direktzugriffsdateien

Für die Verwaltung von Daten in einer Randomdatei sind neben den Befehlen zum Öffnen und Schließen der Datei, die wir bereits besprochen haben, Befehle zum Lesen und Schreiben von Daten in diese Datei notwendig. Da die Werte vor den eigentlichen Schreib- und Lesevorgängen in einen Zwischenspeicher für die zuvor definierten Feldvariablen geschrieben werden (Dateipuffer), sind hierzu weitere Anweisungen erforderlich, die wir an dieser Stelle kurz erläutern wollen.

Beispiel SCHREIBEN:

```
OPEN "TEST.DAT" FOR RANDOM AS #1 LEN=10   'Datei öffnen
FIELD #1, 8 AS Name$, 2 AS Alter$         'Datenfelddefinition
VorName$ = "EGON"                         'Werte für Datensatz
                                          'initialisieren
Alter% = 30
LSET Name$=VorName$                       'Werte linksbündig in
                                          'Dateipuffer schreiben
LSET Alter$=MKI$(Alter%)
PUT #1,1                                  'Datensatz in Datei schreiben
CLOSE #1                                  'Datei schließen
```

Beispiel LESEN:

```
OPEN "TEST.DAT" FOR RANDOM AS #1 LEN=10   'Datei öffnen
FIELD #1, 8 AS Name$, 2 AS Alter$         'Datenfelddefinition
GET #1,1                                  'Datensatz aus Datei lesen
Vorname$=Name$                            'Werte aus Dateipuffer zuweisen
Alter%=CVI(Alter$)
CLOSE #1                                  'Datei schließen
```

Wie Sie an unseren Beispielen erkennen können, wird über *LSET* der Wert in den Dateipuffer linksbündig geschrieben, d.h., der Wert wird in das Datenfeld von links beginnend eingetragen. Alternativ wäre über die Anweisung *RSET* auch ein rechtsbündiges Schreiben möglich. Sowohl *PUT*

und *GET* benötigen die Dateinummer und eine bestimmte Datensatznummer, um Daten schreiben und lesen zu können. In unserem Beispiel arbeiten wir jeweils gezielt mit dem ersten Datensatz.

Werden numerische Daten in den Dateipuffer geschrieben, sollte die Konvertierung in eine Zeichenkette erfolgen. In unserem Beipiel verwenden wir für die Ganzzahl *Alter%* die Funktion *MKI$*. Beim Lesen aus der Datei konvertieren wir die Zeichenkette *Alter$* mit der Funktion *CVI* in eine Ganzzahl zurück. Mit den in der Tabelle 2.4.2 gemachten Angaben können Sie dementsprechend jetzt selbst Variablen eines beliebigen Datentyps in einer Direktzugriffsdatei verwalten. Weitere Hinweise können Sie unserem Praxisteil entnehmen.

Die LOF-Funktion

Mit Hilfe der LOF-Funktion läßt sich die Gesamtdatensatzanzahl einer Direktzugriffsdatei ermitteln. Zwar liefert LOF nur die Dateilänge in Bytes zurück, aber die Datensatzlänge ist uns ja ebenfalls bekannt. Schließlich haben wir diese in der OPEN-Anweisung selbst festgelegt. Mit Hilfe der Integer-Division können wir damit die Datensatzanzahl nach der folgenden Formel errechnen:

```
GesamtDatenSaetze& = LOF(Dateinummer)\DatenSatzlaenge
```

Auslesen von Systemdatum und Systemzeit

Mit Hilfe der in Power Basic integrierten Funktionen *DATE$* und *TIME$* ist es möglich, das Systemdatum bzw. die Systemzeit auszulesen. Wir wollen in unserem Programm HAUSHALT dieses Datum als Buchungsdatum verwenden. Dabei ist allerdings zu berücksichtigen, daß beide Funktionen die amerikanische Schreibweise zurückliefern. Deswegen werden wir im Praxisteil für die Funktion *DATE$* eine Routine entwickeln, um das englische in das deutsche Datumsformat zu übersetzen.

```
Rückgabe DATE$: MM-TT-JJJJ  Bsp.: 11-05-1991

                MM  : Monat
                TT  : Tag
                JJJJ: Jahr

Rückgabe TIME$: HH.MM.SS    Bsp.: 12.30.24

                HH  : Stunde
                MM  : Minute
                SS  : Sekunde
```

Praxis

Da unser Hauptprogramm für diese Sitzung keiner Änderung bedarf, wollen wir zunächst eine Funktion anführen, die das amerikanische Systemdatum in das deutsche Datumsformat übersetzt (Listing 2.4.1). Diese Funktion geben wir am Ende unserer Datei HAUSHALT.BAS in den Editor ein. Sie ermöglicht uns, auf eine Eingabe des Buchungsdatums in unserem Programm zu verzichten.

```
FUNCTION GermanDate$
  GermanDate$=MID$(Date$,4,2) + "." + LEFT$(Date$,2) + "." +_
             RIGHT$(Date$,4)
END FUNCTION
```

Listing 2.4.1: Funktion zum Ermitteln des Systemdatums im deutschen Datumsformat

Mit den Stringfunktionen *MID$, LEFT$* und *RIGHT$* sind umfangreiche Manipulationen von Zeichenketten möglich. Diese nutzen wir, indem wir das amerikanische Systemdatum (Format: MM-TT-JJJJ) in das deutsche Format (Format: TT.MM.JJJJ) umformatieren.

Die Dateiverwaltung

Nun wollen wir uns aber dem Hauptthema dieser Sitzung, der Dateiverwaltung, widmen. Dazu erweitern wir die Routine *DatenEingabe* so, wie in Listing 2.4.2 dargestellt. Die gemachten Änderungen sind durch Fettdruck hervorgehoben.

```
SUB DatenEingabe(DateiNummer, vfarbe%, hfarbe%)
  FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
        4 AS Betrag$
  SatzAnzahl& = LOF(DateiNummer) \ 50
  CALL Hintergrund(vfarbe%, hfarbe%)
  CALL Fenster(8, 10, 60, 8, vfarbe%, hfarbe%)
  CALL InfoZeile("PgUp/PgDn: Datensätzen blättern   Return: Buchung"+_
                 " vornehmen  ESC: Eingabeende", 25, vfarbe%, hfarbe%)
  LOCATE 9, 12                          'Eingabemaske ausgeben
  PRINT "Datensatznummer : ";
  LOCATE 11, 12
  PRINT "Buchungsdatum   : ";
  LOCATE 13, 12
  PRINT "Beschreibung    : ["+SPACE$(36)+"]";
  LOCATE 15, 12
  PRINT "Einnahme/Ausgabe: ["+SPACE$(15)+"]";
  CALL Fenster(19, 10, 60, 3, vfarbe%, hfarbe%)
  LOCATE 20, 12
  PRINT "Eine Einahme wird mit einem positivem [+] und eine Ausgabe"
  LOCATE 21, 12
  PRINT "mit einem negativen Vorzeichen [-] in der ersten Spalte"
```

```
LOCATE 22, 12, 0
PRINT "gekennzeichnet. Die Verwendung des [+] ist dabei wahlfrei."
SatzNummer& = SatzAnzahl&
  IF SatzNummer&>0 THEN
      GET #DateiNummer, SatzNummer&
      Datum$ = BuchungsDatum$
      Kommentar$ = Beschreibung$
      Ein$ = STR$(CVS(Betrag$))
   ELSE
      SatzNummer& = 1
      Datum$ = GermanDate$
END IF
EditFeld% = 0
CALL AktualisiereMaske (SatzNummer&, Datum$, Kommentar$, Ein$, vfarbe%,_
                        hfarbe%)
WHILE a$<>CHR$(27)
  a$ = INKEY$                          'ESC Eingabeende
  b$=RIGHT$(a$, 1)
  IF b$=CHR$(81) THEN                  'PgUp (max. 500 000 Einträge)
        IF SatzNummer& < 500000 AND VAL(Ein$)<>0 THEN
           LSET BuchungsDatum$ = Datum$
           LSET Beschreibung$ = Kommentar$
           LSET Betrag$ = MKS$(VAL(Ein$))
           PUT #Dateinummer, SatzNummer&
           SatzNummer&=SatzNummer& + 1
           IF SatzNummer&>SatzAnzahl& THEN
                Datum$ = GermanDate$
                Kommentar$ = ""
                Ein$ = ""
                SatzAnzahl& = SatzAnzahl& + 1
                ELSE
                GET #Dateinummer, SatzNummer&
                Datum$ = BuchungsDatum$
                Kommentar$ = Beschreibung$
                Ein$ = STR$(CVS(Betrag$))
             END IF
          END IF
        ELSEIF b$=CHR$(73) THEN          'PgDn
          IF SatzNummer&>1 THEN
            LSET BuchungsDatum$ = Datum$
            LSET Beschreibung$ = Kommentar$
            LSET Betrag$ = MKS$(VAL(Ein$))
            PUT #Dateinummer, SatzNummer&
            SatzNummer&=SatzNummer& - 1
            GET #DateiNummer, SatzNummer&
            Datum$ = BuchungsDatum$
            Kommentar$ = Beschreibung$
            Ein$ = STR$(CVS(Betrag$))
          END IF
    ELSEIF a$=CHR$(13) THEN
      EditFeld% = EditFeld% XOR 1     'editiere 1. bzw. 2.Feld
      IF EditFeld%=1 THEN
          Kommentar$ = RTRIM$(Kommentar$)
```

```
          CALL Eingabe("", Kommentar$, 36, 13, 31, vfarbe%, hfarbe%)
        ELSE
          Ein$ = RTRIM$(Ein$)
          CALL Eingabe("R|", Ein$, 15, 15, 31, vfarbe%, hfarbe%)
      END IF
    END IF
    IF b$=CHR$(73) OR b$=CHR$(81) OR a$=CHR$(13) THEN
      CALL AktualisiereMaske (SatzNummer&, Datum$, Kommentar$, Ein$,_
                              vfarbe%, hfarbe%)
    END IF
  WEND
  CALL HinterGrund(vfarbe%, hfarbe%)
END SUB
```

Listing 2.4.2: Erweiterung der Routine DatenEingabe

Zu Beginn der Routine wird über die Anweisung *LOF(DateiNummer)\50* die Gesamtzahl der vorhandenen Datensätze in unserer Arbeitsdatei ermittelt. Diese verwenden wir, um später den letzten Datensatz für unsere Eingabemaske zu aktivieren (*SatzNummer&=SatzAnzahl&*). Da die *LOF*-Funktion lediglich die Gesamtlänge einer Datei in Bytes zurückgibt, wir die Datensatzlänge jedoch kennen, die wir ja schließlich selbst mit 50 Zeichen in der *FIELD*-Anweisung definiert haben, können wir über die Integer-Division (Ergebnis ist eine Ganzzahl) auch die Anzahl der Datensätze errechnen.

Bevor wir die *WHILE-WEND*-Schleife ausführen, überprüfen wir zunächst für die Eingangsmaske, ob bereits Daten in unserer Datei vorhanden sind. Ist dies der Fall (dann *SatzNummer&>0*), so werden die Daten aus der Datei gelesen, ansonsten werden lediglich das Buchungsdatum und die erste Satznummer initialisiert (*SatzNummer&=1 : Datum$=GermanDate$*).

Zwei weitere Funktionen wollen wir in unserem Programm einbauen. Zunächst soll über PgUp in der Datei weitergeblättert werden. Die hierzu erforderlichen Anweisungen sind nachfolgend noch einmal abgedruckt.

```
IF b$=CHR$(81) THEN                          'PgUp (max. 500 000 Einträge)
      'Buchungssumme ungleich Null gefordert!
      IF SatzNummer& < 500000 AND VAL(Ein$)<>0 THEN
        LSET BuchungsDatum$ = Datum$         'Werte in Dateipuffer
        LSET Beschreibung$ = Kommentar$
        LSET Betrag$ = MKS$(VAL(Ein$))
        PUT #Dateinummer, SatzNummer&        'Datensatz schreiben
        SatzNummer&=SatzNummer& + 1          'nächster Datensatz
        IF SatzNummer&>SatzAnzahl& THEN      'neuer Datensatz
            Datum$ = GermanDate$             'Vorgabewerte initialisieren
            Kommentar$ = ""
            Ein$ = ""
```

```
            SatzAnzahl& = SatzAnzahl& + 1   'Satzanzahl aktualisieren
          ELSE                              'Dateiwerte einlesen
            GET #Dateinummer, SatzNummer&
            Datum$ = BuchungsDatum$
            Kommentar$ = Beschreibung$
            Ein$ = STR$(CVS(Betrag$))
        END IF
      END IF
    ELSEIF...
```

Wir wollen in jeder Datei bis zu 500 000 Datensätze verwalten können. Aus diesem Grund ist es notwendig, für die Variable *SatzAnzahl&* den Datentyp Long-Integer zu wählen (Typkennzeichen: &). Um die Erzeugung eines neuen Datensatzes bei jedem Betätigen der [PgUp]-Taste zu vermeiden, kontrollieren wir zunächst immer, ob überhaupt ein Buchungsbetrag eingegeben worden ist. Sollte das nicht der Fall sein, ist das Speichern der Daten auch nicht sinnvoll. Außerdem müssen wir bei jedem Blättern kontrollieren, ob ein neuer Datensatz angelegt wird (*SatzNummer&>SatzAnzahl*), damit die Maskenwerte, wie bereits zu Beginn, für den neuen Datensatz korrekt initialisiert werden können.

⚠ **Beachten Sie, daß nach dem Speichern eines neuen Datensatzes (dann *SatzNummer&>SatzAnzahl&*) durch die Anweisung *SatzAnzahl&=SatzAnzahl&+1*, die Gesamtanzahl der Datensätze aktualisiert wird. Die am Routinenanfang ermittelte Datensatzanzahl wird nämlich in diesem Fall durch das Weiterblättern erhöht. Die korrekte *SatzAnzahl&* benötigen wir intern als Kontrollvariable für die Eingabemaske, so daß eine korrekte Kenntnis der Datensatzanzahl unbedingt erforderlich ist.**

Das Zurückblättern in der Datei verläuft ähnlich wie das Weiterblättern. Dabei brauchen wir lediglich darauf zu achten, daß nicht vor den ersten Datensatz geblättert werden darf. Die Variable *SatzNummer&* muß vor dem Blättern also einen Wert **größer** 1 haben, da dieser innerhalb dieses Blocks um 1 abgemindert wird und nicht zu 0 werden darf!

Da wir für den Buchungsbetrag eine einfachgenaue Fließkommazahl gewählt haben, benötigen wir für die Verwaltung der Daten in der Direktzugriffsdatei die Konvertierroutinen *MKS$* und *CVS* (siehe Tabelle 2.4.2). Da sämtliche Eingaben als Zeichenketten erfolgen (Routine *Eingabe*), setzen wir zudem die Funktion *STR$* ein, um den aus dem Dateipuffer ermittelten numerischen Wert als Zeichenkette an unsere Eingaberoutine übergeben zu können.

Nachdem Sie die Routine *DatenEingabe* komplett im Editor überarbeitet haben, können Sie das Programm ausführen und testen ([Alt] + [F9] [F9]). Sie werden sehen, daß Eingaben beim Blättern nicht verlorengehen,

sondern wieder in der Bildschirmmaske erscheinen. Dafür ist unsere im Hintergrund arbeitende Dateiverwaltung verantwortlich (Bild 2.4.1). Sie sehen also, daß die Realisierung einer Dateiverwaltung gar nicht so schwer ist.

Mit den Grundlagen, die Sie bislang erlernt haben, sollten Sie nunmehr in der Lage sein, eigene Dateiverwaltungen (z.B. eine Adreßverwaltung) zu entwickeln. In unserem Programm haben wir damit die Entwicklung der Programmoberfläche und der Dateiverwaltung abgeschlossen. Damit können wir uns nun mit der Datenauswertung befassen, bevor wir uns der weiteren Programmoptimierung zuwenden.

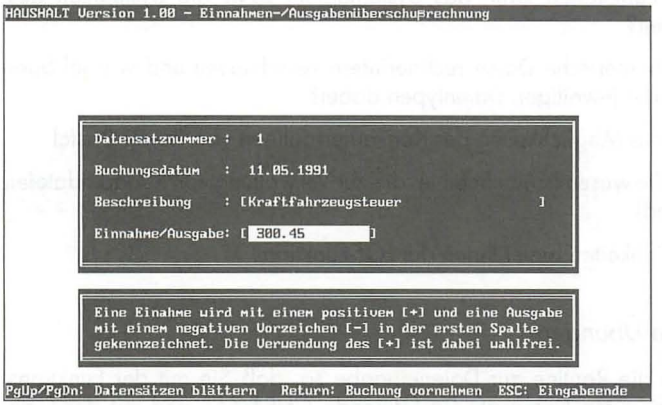

Bild 2.4.1: Eingaberoutine mit aktiviertem Eingabefeld

So, wie wir unsere Routine bislang angelegt haben, können die Daten von beliebigen Zeiträumen eingegeben werden. Bei der Auswertung werden wir uns allerdings immer auf ein spezielles Buchungsjahr beziehen. Dieses beginnt jeweils am 1. Januar und endet am 31. Dezember.

In der derzeitigen Fassung der Routine wird das Buchungsdatum automatisch gesetzt. Wenn Sie ein spezielles Datum für den Buchungssatz benötigen, sollten Sie es in das Kommentarfeld eintragen. Bedenken Sie aber, daß Sie aufgrund der Einschränkungen unserer Eingaberoutine höchstens einen Dezimalpunkt eingeben können. Ein Datum in der Form »11.05.1991« sollten Sie z.B. in der Form »11/5/91« eingeben.

Wenn die Eingabemaske nach dem Aufruf des Menüpunktes *Dateneingabe* auf dem Bildschirm erscheint, ist zunächst noch kein spezielles Eingabefeld aktiviert. Durch [Return] können Sie eine Eingabe für den Kommentar vornehmen. Beachten Sie, daß das aktiv geschaltete Editierfeld

farblich hervorgehoben wird. Durch erneutes Drücken von [Return] verlassen Sie das Eingabefeld wieder. Wiederholen Sie diesen Vorgang, wird das nächste Mal das Eingabefeld für den Buchungsbetrag aktiviert. Sie können diese Feldumschaltung optimieren, indem Sie die Cursorsteuerung so berücksichtigen, daß damit eine unmittelbare Feldumschaltung möglich wird.

Workshop

Checkliste

1. Mit welchen Funktionen kann das Systemdatum bzw. die Systemzeit ausgelesen werden?
2. Wie werden numerische Daten rechnerintern verschlüsselt und wieviel Speicher belegen die jeweiligen Datentypen dabei?
3. Erläutern Sie die Möglichkeiten der Konvertierroutinen von Power Basic!
4. Erläutern Sie die wesentlichen Befehle, die zur Verwaltung von Randomdateien notwendig sind!
5. Welche Möglichkeiten bietet Ihnen die *LOF*-Funktion?

Ideen für eigene Übungen

- Erweitern Sie die Routine zur Dateneingabe so, daß Sie mit der Funktionstaste [F7] den ersten Datensatz (Tastaturcode *CHR$(65)*) und mit der Funktionstaste [F8] (Tastaturcode *CHR$(66)*) den letzten laden.
- Erweitern Sie den Aufbau des Hauptbildschirms so, daß das aktuelle Datum und die aktuelle Zeit mit auf dem Bildschirm ausgegeben wird. Verwenden Sie dazu das jeweilige deutsche Format!

2.5 Fünfte Sitzung: Datenauswertung

Die Themen dieser Sitzung:

- Mathematische Funktionen
- Stringfunktionen
- Dateiauswertung

Information

Die Buchungssätze, die wir in unsere Arbeitsdatei geschrieben haben, müssen für die Berechnung des Einnahmen-/Ausgabenüberschusses miteinander verknüpft werden. Dies wollen wir in dieser Sitzung realisieren. Dazu werden wir neben internen Power-Basic-Funktionen noch einmal auf die Dateiverwaltung und einfache Auswertungsmöglichkeiten zu sprechen kommen.

Bei der Ausgabe beschränken wir uns zunächst auf eine Tabellenausgabe auf dem Bildschirm, wobei wir allerdings bereits berücksichtigen wollen, daß die eigentliche Auswertungsroutine auch für die Druckerausgabe bzw. Grafikausgabe auf dem Bildschirm benötigt wird. Die Zahlenwerte sollen dabei auf zwei Nachkommastellen gerundet werden. Da keine brauchbare Rundungsfunktion in Power Basic implementiert ist, werden wir uns hierzu eine eigene Funktion programmieren.

Mathematische Funktionen

Power Basic verfügt über die wichtigsten Grundfunktionen, die für mathematische Berechnungen benötigt werden. Dadurch können in Verbindung mit den mathematischen Operatoren, die wir bereits kennengelernt haben, auch komplizierte mathematische Aufgabenstellungen gelöst werden. Eine Zusammenstellung der Funktionen findet sich in Tabelle 2.5.1. Dabei ist zu beachten, daß sämtliche Winkelfunktionen (*ATN, COS, SIN, TAN*) das übergebene Argument im Bogenmaß (Rad) erwarten. Diese Umrechnung werden wir im Grafikkapitel genauer behandeln.

ABS	Absolutbetrag eines Argumentes
ATN	Arcustangens eines Argumentes
COS	Cosinus eines Argumentes
EXP	e hoch Argument
EXP2	2 hoch Argument
EXP10	10 hoch Argument
LOG	natürlicher Logarithmus (Basis e)
LOG2	Logarithmus zur Basis 2
LOG10	Logarithmus zur Basis 10
SGN	Vorzeichen eines Argumentes
SIN	Sinus eines Argumentes
SQR	Quadratwurzel eines Elementes
TAN	Tangens eines Elementes

Tabelle 2.5.1: Liste der mathematischen Funktionen

Um den Einsatz der mathematischen Funktionen zu veranschaulichen, wollen wir an dieser Stelle den Aufruf allgemein und an einigen kleinen Beispielen anführen.

Allgemein:

```
Ergebnis = mathematische Funktion (Argument)
```

Beispiele:

```
y% = ABS(-3)
y! = SIN(0.35)
y# = COS(0.253)
```

Mit Hilfe der vorhandenen Funktionen können alle weiteren mathematischen Funktionen formuliert werden. Benötigen wir zum Beispiel die Funktion für den Sinushyperbolicus, so kann recht schnell durch einen Blick in eine Formelsammlung eine Umrechnungsmöglichkeit gefunden werden, die in diesem Fall *SINH = (EXP(arg) – EXP(-arg) / 2* lautet. Damit können wir recht schnell eine Funktion programmieren.

```
FUNCTION SINH!(X!)
  SINH! = (EXP(X!)-EXP(-X!))/2
END FUNCTION
```

Unser Beispiel arbeitet in diesem Fall mit einfachgenauen Fließkommawerten. Diese benutzerdefinierte, mathematische Funktion können wir nun wie eine interne Basic-Funktion nutzen. Um die neue Anweisung in unseren Programmen zugänglich zu machen, können wir sie entweder in den Quelltext unseres Programms schreiben, sie durch einen *$INCLUDE*-Befehl in unser Hauptprogramm einbinden lassen (falls die Funktion in einer gesonderten Quelldatei vorliegt) oder wir integrieren sie in eine Unit und

binden diese in unser Hauptprogramm mit *$LINK* ein. Dann müssen wir am Programmanfang allerdings die *DECLARE*-Anweisung *DECLARE FUNCTION SINH!(SINGLE)* einfügen.

Stringfunktionen

Im Vergleich zu anderen Basic-Dialekten bietet Power Basic nicht nur einen recht großen Umfang an mathematischen Funktionen, sondern auch umfangreiche Stringfunktionen an. Die Zusammenstellung ist in Tabelle 2.5.2 enthalten.

EXTRACT$	Teilstring abspalten, der auf spezielle Zeichen hinweist
INSTR	liefert die Startposition eines Suchstrings in einem String
LCASE$	Umwandlung Groß- in Kleinschrift
LEFT$	liefert linke Teilzeichenkette zurück
LTRIM$	Herausfiltern von Zeichen aus einer Zeichenkette (von links)
MID$	mittleren Teil eines Strings ermitteln
REMOVE$	Zeichen aus einem String entfernen
REPEAT$	gibt String zurück, der einen String n-mal beinhaltet
REPLACE	Teile eines Strings ersetzen
RIGHT$	liefert rechte Teilzeichenkette zurück
RTRIM$	Herausfiltern von Zeichen aus einem String (von rechts)
SPACE$	liefert String mit n Leerzeichen zurück
STRING$	String erzeugen, der ein bestimmtes Zeichen n-mal enthält
TALLY	Anzahl von Zeichen und Strings im Gesamtstring ermitteln
UCASE$	Umwandlung Klein- in Großschrift
USING$	Zeichenkette formatieren
VERIFY	spezieller Stringvergleich

Tabelle 2.5.2: Stringfunktionen von Power Basic

Die wichtigsten Funktionen *MID$, LEFT$* und *RIGHT$* haben wir bereits für die Routine *GermanDate$* genutzt. Anders als bei anderen Basic-Dialekten werden bei den Funktionen *UCASE$* und *LCASE$* auch deutsche Umlaute berücksichtigt.

Power Basic arbeitet gegenüber anderen Programmiersprachen, wie z.B. Pascal, mit dynamischen Zeichenketten. Zeichenketten können also variabel in ihrer Länge sein und aus 0 bis maximal 32750 Zeichen bestehen.

☞ Beachten Sie, daß über das Power-Basic-Menü bzw. über den Kompilier-Befehl *$STRING* eine Reduzierung der maximalen Stringlänge auf bis zu 1006 Zeichen möglich ist.

Neben den Stringfunktionen, die in der Tabelle 2.5.2 aufgeführt sind, spielt die Verkettung von Zeichenketten in der Programmierung eine wesentliche Rolle. Diese erfolgt über den Operator +. Um z.B. den Vornamen "*Harry*" mit dem Nachnamen "*Meier*" in einem einzelnen String zu vereinen, könnten Sie folgende Anweisung verwenden:

```
Name$ = "Harry" + " " + "Meier"
```

Dabei ist es natürlich sinnvoll, die beiden Teilnamen durch ein Leerzeichen zu trennen. Dieselbe Anweisung könnte aber auch durch die folgenden Befehle realisiert werden:

```
Name$ = "Harry" + SPACE$(1) + "Meier"
Name$ = "Harry" + CHR$(32) + "Meier"
Name$ = "Harry" + STRING$(1," ") + "Meier"
```

An diesem Beispiel können Sie sehen, daß es für eine Formulierung in der Regel mehrere Lösungsmöglichkeiten gibt. Sie werden sich sicherlich denken können, wenn bei einem einzelnen Befehl bereits mehrere Formulierungen möglich sind, daß bei größeren Programmen eine unterschiedliche Realisierung mit dem gleichen Ergebnis erst recht wahrscheinlich ist. Es ist also durchaus möglich, daß unterschiedliche Programmierer für ein- und dasselbe Problem unterschiedliche Lösungswege einschlagen, die sich im Endeffekt nicht im Ergebnis, sondern z.B. in der Ausführungsgeschwindigkeit unterscheiden. Die Anwendung der Stringfunktionen haben Sie bereits in den ersten Kapiteln ausgiebig eingesetzt. Auch bei der Datenauswertung können wir darauf nicht verzichten.

Datenauswertung

Um Daten aus einer Datei auswerten zu können, müssen wir natürlich genau wissen, um welchen Dateityp es sich handelt und welche Datenstruktur vorhanden ist. In der Regel ist es dann ausreichend, die Daten zu lesen und die Ergebnisse im Speicher zu halten, um sie entweder auf dem Bildschirm, dem Drucker oder in einer Bildschirmgrafik auszugeben. Die einfachste Realisierung ergibt sich, wenn man die Datei vom Dateianfang bis zum Dateiende in einer Schleife liest und die benötigten Werte unmittelbar verarbeitet. Auf das Schreiben der Ergebnisse in eine spezielle Ergebnisdatei wollen wir an dieser Stelle verzichten.

Da wir in unserem Programm mit einer Direktzugriffsdatei arbeiten, ist es sinnvoll, folgende Schleife für die Auswertung einzusetzen:

Auswertung einer Direktzugriffsdatei:

```
FOR Datensatz&=1 TO LOF(DateiNummer)\SatzLaenge%
   GET #DateiNummer, Datensatz
   'Anweisungen zur Auswertung
NEXT Datensatz&
```

Auswertung einer sequentiellen Datei:

```
WHILE NOT EOF(DateiNummer)
  LINE INPUT #DateiNummer, Text$
  'Anweisungen zur Auswertung
WEND
```

Bei sequentiellen Dateien ist das Dateiende durch eine End-Of-File-Marke gekennzeichnet, die durch die *EOF*-Funktion abgefragt werden kann. Diese Funktion ermöglicht genau bis zum Dateiende zu lesen. Erinnern Sie sich daran, daß jeder »Datensatz« in einer sequentiellen Datei eine unterschiedliche Länge haben kann und sich somit eine Ermittlung der Datensatzanzahl innerhalb dieser Datei nicht wie bei der Randomdatei realisieren läßt. Ein Beispiel für die Auswertung einer sequentiellen Datei werden wir in unserer neunten Sitzung besprechen.

Praxis

Unsere Arbeitsdatei ist eine Direktzugriffsdatei mit bereits festgelegten Datenfeldern. Da unsere Datenauswertung über das Hauptauswahlmenü unseres Programms HAUSHALT aufrufbar sein soll, müssen wir dafür sorgen, daß bereits eine Arbeitsdatei gewählt ist. Dazu können wir wieder unsere Routine *KeineDateiAktiv* verwenden, die wir im Hauptprogramm

(Listing 2.5.1) in der *SELECT-CASE*-Auswahl für die Datenauswertung integrieren.

```
: 'unverändertes Hauptprogramm
:
CASE 3      'Datenauswertung
  IF ArbeitsDatei$="keine" THEN
     CALL KeineDateiAktiv(vfarbe%, hfarbe%)
   ELSE     'erster Parameter Mode%
     CALL DatenAuswertung(1, DNr, vfarbe%, hfarbe%)
  END IF
CASE 4      'Ergebnisse ausdrucken
:
: 'unverändertes Hauptprogramm
```

Listing 2.5.1: Angepaßtes Hauptprogramm für die Datenauswertung auf dem Bildschirm

Unsere Routine *DatenAuswertung* soll nicht nur die Einnahmen-/Ausgabenüberschußrechnung durchführen, sie soll auch über den Parameter *Mode%* den Aufruf der Drucker- bzw. der Grafikroutinen übernehmen. Den Parameter werden wir in dieser Sitzung noch nicht nutzen, er erhält erst im weiteren Verlauf unseres Tutoriums eine Bedeutung. Sie sollten allerdings schon berücksichtigen, daß *Mode%=1* für die Bildschirmausgabe, *Mode%=2* für die Druckerausgabe und sonstige Werte für die Grafikausgabe reserviert werden sollen. Das erklärt auch den Aufruf der Routine *DatenAuswertung* in unserem Hauptprogramm. Nachdem Sie das Hauptprogramm erweitert haben, können die Unterprogramme und Funktionen für die Datenauswertung in unserem Hauptprogramm am Dateiende formuliert werden. Dazu bewegen Sie den Textcursor im Editor auf das Textende der Datei HAUSHALT.BAS und geben die Funktion *Runden!* und das Unterprogramm *DatenAuswertung* ein.

Die Datenauswertung

Zunächst müssen wir die Ergebnisse festlegen, die unsere Routine ermitteln soll. Wir wollen die Gesamteinnahmen und die Gesamtausgaben für ein bestimmtes Buchungsjahr ermitteln. Daraus können wir direkt den Gesamtsaldo, also die Differenz zwischen Einnahmen und Ausgaben, errechnen. Außerdem wollen wir für jeden Monat eines Jahres einen Gesamtsaldo ermitteln, um Schwankungen innerhalb des Jahres erkennbar zu machen. So kann im Monat Januar durchaus eine negative Bilanz und im Monat Februar eine positive Bilanz vorliegen. Diese Ergebnisse wollen wir später auch grafisch veranschaulichen. Unsere Werte sollen auf zwei Zahlen nach dem Komma gerundet werden. Daher werden wir zunächst eine Funktion schreiben, die für uns das Runden übernimmt (Listing 2.5.2).

Alternativ könnten wir an dieser Stelle auch die interne Power-Basic-Funktion *ROUND* einsetzen.

```
FUNCTION Runden!(Zahl!)
  Runden! = INT(Zahl! * 100 + 0.5) / 100
END FUNCTION
```

Listing 2.5.2: Die Funktion Runden!

Wie Sie sehen, beschränken wir uns wieder auf Fließkommazahlen mit einfacher Genauigkeit. Den Datentyp also, der auch in unserer Dateiverwaltung verwendet wird, um die Buchungsbeträge zu speichern. Die hier verwendete Power-Basic-Funktion *INT* gehört zu den Konvertierfunktionen und wandelt den Ausdruck (Zahl!*100+0.5) in eine Ganzzahl (Integer) um.

Nun wollen wir aber das Unterprogramm für die Datenauswertung mit Ausgabe der Ergebnisse in einem Bildschirmfenster vorstellen. Die Routinen für Drucker- und Grafikausgabe werden später an geeigneten Stellen integriert.

```
SUB DatenAuswertung(Mode%, Dateinummer, vfarbe%, hfarbe%)
  'Mode% reserviert für spätere Fassung
  FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
        4 AS Betrag$
  CALL Hintergrund(vfarbe%, hfarbe%)
  CALL Fenster(10, 16, 46, 1, vfarbe%, hfarbe%)
  Jahr$ = RTRIM$(RIGHT$(Date$, 4))
  CALL Eingabe("I| Bildschirmauswertung für welches Jahr: ", Jahr$,_
               4, 11, 18, vfarbe%, hfarbe%)
  IF LEN(Jahr$)<> 4 THEN EXIT SUB
  LOCATE ,,0
  FOR SatzNummer&=1 TO LOF(Dateinummer)\50
    GET #Dateinummer, SatzNummer&
    Datum$ = BuchungsDatum$
    Kommentar$ = Beschreibung$
    Betrag! = CVS(Betrag$)
    IF RIGHT$(Datum$, 4) = Jahr$ THEN
      IF Betrag! < 0 THEN                        'Gesamtsummen
          Ausgabe! = Ausgabe! + ABS(Betrag!)
        ELSEIF Betrag! > 0 THEN
          Einnahme! = Einnahme! + Betrag!
      END IF
      SELECT CASE MID$(Datum$, 4, 2)             'monatlicher Saldo
        CASE "01"                  'Januar
          Januar! = Januar! + Betrag!
        CASE "02"                  'Februar
          Februar! = Februar! + Betrag!
        CASE "03"                  'März
          Maerz! = Maerz! + Betrag!
        CASE "04"                  'April
```

```
        April! = April! + Betrag!
      CASE "05"                 'Mai
        Mai! = Mai! + Betrag!
      CASE "06"                 'Juni
        Juni! = Juni! + Betrag!
      CASE "07"                 'Juli
        Juli! = Juli! + Betrag!
      CASE "08"                 'August
        August! = August! + Betrag!
      CASE "09"                 'September
        September! = September! + Betrag!
      CASE "10"                 'Oktober
        Oktober! = Oktober! + Betrag!
      CASE "11"                 'November
        November! = November! + Betrag!
      CASE "12"                 'Dezember
        Dezember! = Dezember! + Betrag!
      CASE ELSE
    END SELECT
  END IF
NEXT SatzNummer&
Gesamt! = Einnahme! - Ausgabe!
CALL Fenster(5, 5, 70, 15, vfarbe%, hfarbe%)
Gesamt! = Einnahme! - Ausgabe!
LOCATE  6, 8: PRINT "Auswertungsjahr.............:  ";Jahr$
LOCATE  7, 8: PRINT "Summe aller Einnahmen.......: ";Runden!(Einnahme!)
LOCATE  8, 8: PRINT "Summe aller Ausgaben........: ";Runden!(Ausgabe!)
LOCATE  9, 8: PRINT "Gesamtsaldo.................: ";Runden!(Gesamt!)
LOCATE 10, 5: PRINT "├" + STRING$(70, "-") + "┤";
LOCATE 12, 8: PRINT "Saldo Januar...:";Runden!(Januar!)
LOCATE 12,45: PRINT "Juli...........:";Runden!(Juli!)
LOCATE 13, 8: PRINT "      Februar..:";Runden!(Februar!)
LOCATE 13,45: PRINT "August.........:";Runden!(August!)
LOCATE 14, 8: PRINT "      März.....:";Runden!(Maerz!)
LOCATE 14,45: PRINT "September......:";Runden!(September!)
LOCATE 15, 8: PRINT "      April....:";Runden!(April!)
LOCATE 15,45: PRINT "Oktober........:";Runden!(Oktober!)
LOCATE 16, 8: PRINT "      Mai......:";Runden!(Mai!)
LOCATE 16,45: PRINT "November.......:";Runden!(November!)
LOCATE 17, 8: PRINT "      Juni.....:";Runden!(Juni!)
LOCATE 17,45: PRINT "Dezember.......:";Runden!(Dezember!)
LOCATE ,,0
CALL Warte
CALL Hintergrund(vfarbe%, hfarbe%)
END SUB
```

Listing 2.5.3: Datenauswertung mit Bildschirmausgabe

Die Auswertung erfolgt in mehreren wesentlichen Schritten. Zunächst wird über die Routine das Jahr der Buchungsauswertung abgefragt (Bild 2.5.1). Für die Eingabefeldvorgabe der Zeichenkette wird durch die Einleitung "*I*|" gesichert, daß nur numerische Zeichen eingebbar sind.

Bild 2.5.1: Abfrage des Buchungsjahrs für die Datenauswertung im Programm HAUSHALT

☞ Beachten Sie, daß allerdings die Zeichen »+« und »–« ebenfalls in der ersten Spalte eingegeben werden können, was in diesem Fall nicht sinnvoll ist. Es wäre allerdings möglich, die Eingaberoutine so zu erweitern, daß auch spezielle Datums- und Jahresformate eingebbar wären.

In unserer Auswertung kontrollieren wir nur, ob die Zeichenkette für das Jahr vier Zeichen enthält (dann *LEN(Jahr$)= 4*). Ist dies nicht der Fall (dann *LEN(Jahr$)<>4*) wird das Unterprogramm über die Anweisung *EXIT SUB* verlassen. Da es kaum realistisch ist, daß für das Jahr »999« Buchungseinträge vorhanden sind, wäre es unsinnig, die gesamte Datei nach passenden Buchungssätzen für dieses Jahr zu durchsuchen.

☞ Theoretisch könnte man auch den Wert von *Jahr$* kontrollieren und damit ein genaues Grenzjahr, ab dem Buchungen zulässig sind, festlegen. So könnte man z.B. das Erscheinungsdatum unserer Software für diese Definition heranziehen. Wollten wir z.B. nur Buchungszeiträume ab 1990 berücksichtigen, könnte die Anweisung *IF VAL(Jahr$)<1990 THEN EXIT SUB* lauten.

Im Anschluß verwenden wir die bereits im Informationsteil zu dieser Sitzung erläuterte *FOR-NEXT*-Schleife, um die Arbeitsdatei satzweise zu lesen und auszuwerten. Dabei werden nur Datensätze für das angegebene Buchungsjahr berücksichtigt (*IF RIGHT$(Datum$,4)=Jahr$ THEN...*). Um die Gesamtausgaben und Gesamteinnahmen zu ermitteln, brauchen wir von den zu berücksichtigenden Buchungseinträgen lediglich überprüfen, ob es sich um positive oder negative Werte handelt. Negative Werte sind Ausgaben und werden über die Anweisung *Ausgabe!=Ausgabe!+ABS(Betrag!)* addiert. Die Summe aller Ausgaben ist dabei durch Einsatz der Absolutfunktion *ABS* positiv.

⚠ ***Ausgabe!* und *Einnahme!* sind lokale und nicht statische Variablen. Daher sind Ihre Anfangswerte beim Aufruf der Funktion jeweils 0. Lokal heißt, die Variablen sind nur in dem Unterprogramm *DatenAuswertung* bekannt. Statisch heißt, daß die Endwerte der Variablen, nach dem ersten Aufruf der Routine, bei einem eventuellen zweiten Aufruf noch die Werte enthalten, die sich durch den ersten Aufruf ergeben haben. Wäre das der Fall, müßten *Ausgabe!* und *Einnahme!* am Beginn des Unterprogramms mit 0 initialisiert werden.**

Dementsprechend werden alle positiven Buchungseinträge in der Variablen *Einnahme!* addiert. Diese mathematischen Zuweisungen sollten Sie sich noch einmal genauer anschauen.

```
Einnahme! = Einnahme! + Betrag!
```

Durch diese Anweisung wird der Variablen *Einnahme!* ein Wert zugewiesen, der aus der Summe des alten Werts der Variablen *Einnahme!* und dem neu eingelesenen Wert *Betrag!* besteht. Durch den Einsatz innerhalb der *FOR-NEXT*-Schleife erreichen wir auf einfachste Art und Weise die Summation der passenden Werte.

Die monatliche Saldenbildung muß dementsprechend ebenfalls in der *FOR-NEXT*-Schleife erfolgen. Die entsprechenden Zuweisungen stehen dabei innerhalb einer *SELECT-CASE*-Auswahl. Dazu haben wir zwölf Variablen für zwölf Monate eingeführt.

☞ Eine Alternativlösung hierzu sollen Sie im Übungsteil eigenständig unter Verwendung eines Datenfeldes realisieren.

Um eine Zuordnung zu ermöglichen, ermitteln wir über *MID$(Datum$,4,2)* den Monat aus dem Buchungsdatum des jeweiligen Datensatzes. Die Zuweisung erfolgt ähnlich wie bereits bei den Gesamtsummen. Positive und negative werden bei der Summenbildung automatisch verrechnet.

Damit ist die Datenauswertung bereits erfolgt. Natürlich sollen die Ergebnisse für den Anwender auch sichtbar werden. Daher folgen nach der Auswertung unmittelbar die Anweisungen für die Bildschirmausgabe, wobei auch die benutzerdefinierte Funktion *Runden!* zum Einsatz kommt. Das Bildschirmfenster bleibt so lange auf dem Bildschirm, bis eine beliebige Taste gedrückt wird (*CALL Warte*).

```
HAUSHALT Version 1.00 - Einnahmen-/Ausgabenüberschußrechnung

  Auswertungsjahr.............:  1991
  Summe aller Einnahmen.......:  4794.95
  Summe aller Ausgaben........:  565.99
  Gesamtsaldo.................:  4228.96

  Saldo Januar...: 0                Juli...........:-233
        Februar..: 0                August.........: 0
        März.....: 0                September......: 0
        April....: 0                Oktober........: 0
        Mai......: 1027.96          November.......: 0
        Juni.....: 3434             Dezember.......: 0

↑↓: Anwahl Menüeintrag   RETURN: Menüauswahl   Arbeitsdatei: 1991.DAT
```

Bild 2.5.2: Bildschirmausgabe der Datenauswertung im Programm HAUSHALT

☞ Die *LOCATE*-Funktion ermöglicht es, mit der Anweisung *LOCATE ,,0* den Textcursor unsichtbar und mit *LOCATE ,,1* sichtbar zu machen.

Nun können wir uns der Ausgabe auf dem Drucker und der grafischen Ausgabe zuwenden.

Workshop

Checkliste

1. Über welche mathematische Funktionen verfügt Power Basic?
2. Wie können in Power Basic nicht vorhandene mathematische Funktionen realisiert werden?
3. Welche wichtigen Stringfunktionen von Power Basic haben Sie kennengelernt?
4. Wie können Daten aus einer Datei besonders einfach ausgewertet werden?

Ideen für eigene Übungen

- Schreiben Sie eine Funktion, die die Fläche in einem Quadrat berechnet.
- Schreiben Sie eine Funktion, die die Fläche eines Kreises berechnet.
- Schreiben Sie eine eigene Stringfunktion, die ein bestimmtes Wort einer Zeichenkette, die aus mehreren Wörtern besteht, zurückgibt. Nehmen Sie z.B. die Zeichenkette "*Es ist ein schöner Tag*", dann muß beim Ermitteln z.B. des dritten Wortes, die Zeichenkette "*ein*" zurückgegeben werden.

- In der derzeitigen Fassung der Routine *DatenAuswertung* wird für jeden Monatssaldo eine eigene Variable verwendet. Wie ließe sich dies über ein Datenfeld verbessern?
- Ändern Sie die Routine *DatenAuswertung* so ab, daß eine Auswertung über bestimmte Buchungszeiträume möglich ist (z.B. 1.03.91 bis 1.03.93). Überlegen Sie dabei, ob es sinnvoll ist, die einzelnen Jahre zunächst getrennt auszuwerten, um im Anschluß eine Gesamtstatistik zu erstellen.
- Entwickeln Sie ein Konzept, das es ermöglicht, mehrere Dateien unseres Haushaltsprogramms für die Datenauswertung zu verknüpfen! In diesem Fall könnten z.B. Geschäftsverbindungen zu bestimmten Personen oder Firmen in getrennten Dateien verwaltet und bei Bedarf verknüpft werden.

2.6 Sechste Sitzung: Druckerausgabe – Daten schwarz auf weiß

Die Themen dieser Sitzung:

- Anweisungen zur Druckerausgabe
- Aufbereitung von Daten für die Druckerausgabe
- Erstellung eines Seitenkopfes für die Druckerausgabe
- Verwendung von Drucker-Steuercodes
- Statische Variablen

Information

In dieser Sitzung wollen wir uns mit der Druckerausgabe befassen. Sämtliche Buchungssätze der jeweils aktiven Arbeitsdatei in unserem Programm HAUSHALT sollen in Tabellenform auf dem Drucker ausgegeben werden. Dabei ist zunächst zu berücksichtigen, daß die Druckerausgabe selbst über mehrere Textseiten gehen kann, da insgesamt bis zu 500 000 Datensätze in einer Datei stehen dürfen. Die einzelnen Seiten sollen ordnungshalber mit einem einheitlichen Kopf und einer Seitennumerierung versehen werden. Auf die Ausgabe der monatlichen Gesamtsalden werden wir an dieser Stelle verzichten, lediglich die Gesamtsummen der Einnahmen und der Ausgaben sowie der Gesamtsaldo eines Buchungsjahres sollen am Tabellenende ausgegeben werden.

Anweisungen zur Druckerausgabe

Power Basic stellt mit *LPRINT* eine einfache Anweisung zur Verfügung, die es erlaubt, Zeichen zum Drucker zu schicken. Dabei wird immer die erste parallele Schnittstelle eingesetzt. Durch *LPRINT USING* sind auch formatierte Ausgaben auf dem Drucker möglich.

Beispiel:

```
a%=3
a!=3.5762
LPRINT "Dies ist eine Druckerausgabe"
```

```
LPRINT "Variable a%: "; a%
LPRINT USING "formatiert: ###,##";a!
LPRINT "eins+";
LPRINT "zwei"
```

Druckerausgabe:

```
Dies ist eine Druckerausgabe
Variable a%: 3
formatiert:   3,58
eins+zwei
```

Wie Sie an unserem Beispiel erkennen, kann der Zeilenvorschub des Druckers unterdrückt werden, wenn die *LPRINT*-Anweisung mit einem Semikolon endet. Der Zeilenvorschub erfolgt wie bei der *PRINT*-Anweisung automatisch, wenn das Semikolon am Ende der Anweisung weggelassen wird. Die Standardzeilenbreite des Druckers ist auf 80 Zeichen voreingestellt. Verfügen Sie über einen Drucker, der auch DIN-A3-Formate bedrucken kann, oder Sie wollen eine kleine Schrift (z.B. LinePrinter) auf Ihrem Drucker verwenden, wodurch mehr als 80 Zeichen auf eine DIN-A4-Seite in eine Textzeile passen, so müssen Sie zunächst den *WIDTH*-Befehl verwenden. Ist Ihr Drucker an der Schnittstelle LPT1 angeschlossen, was der Regelfall ist, so lautet die Anweisung *WIDTH "LPT1:", 255*. In diesem Fall wird **kein** automatischer Zeilenvorschub mehr zum Drucker gesendet, es sei denn, Sie erzwingen einen Vorschub, indem Sie *LPRINT* ohne abschließendes Semikolon verwenden.

⚠ **Auch wenn Sie nie *LPRINT* ohne abschließendes Semikolon einsetzen, wird standardmäßig nach genau 80 Zeichen ein Zeilenvorschub durchgeführt. Erst mit der *WIDTH*-Anweisung können Textzeilen mit mehr als 80 Zeichen gedruckt werden.**

Um die Zeilenbreite z.B. auf 150 Zeichen festzulegen, verwenden Sie die Anweisung *WIDTH "LPT1:", 150*. Ist Ihr Drucker an einer anderen Schnittstelle angeschlossen, ersetzen Sie *LPT1* durch *LPT2* bei der zweiten, und durch *LPT3* bei der dritten parallelen Schnittstelle.

Sollte Ihr Drucker allerdings nicht an der ersten parallelen Schnittstelle angeschlossen sein, müssen Sie auf die *LPRINT*-Anweisung verzichten. Aber lassen Sie sich beruhigen, Power Basic bietet eine Anweisung, die weitaus flexibler ist als *LPRINT*. Wir werden zwar in unserem Programm *LPRINT* verwenden, aber damit Sie das Programm HAUSHALT an Ihren Rechner anpassen können, wollen wir Ihnen diese Lösungsmöglichkeit nicht vorenthalten.

Mit der Anweisung *OPEN "LPT1:" AS #1: WIDTH #1, 255* öffnen Sie den Drucker als Datei und unterdrücken anschließend mit *WIDTH* den automatischen Zeilenvorschub. Um nun auf den Drucker zu schreiben, verwenden Sie den *PRINT#*-Befehl. Um z.B. *"Hallo"* zu drucken, benutzen Sie die Anweisung *PRINT #1, "Hallo"*. Für die zweite parallele Schnittstelle würden die Anweisungen wie folgt lauten:

Druckerausgabe (zweite parallele Schnittstelle):

```
OPEN "LPT2" AS #1      'LPT2 öffnen
WIDTH #1, 255          'automatischen Zeilenvorschub abschalten
PRINT #1, "Hallo"      'auf LPT2 drucken
CLOSE #1               'LPT2 schließen
```

Sind die Ausgaben eines Programms für Bildschirm, Datei und Drucker identisch, so können Sie über die *OPEN*-Anweisung, durch unterschiedliches Öffnen, dieselben *PRINT#*-Anweisungen verwenden. Zur Veranschaulichung soll Ihnen das nächste Beispiel dienen.

Beispiel:

```
OPEN "SCRN:" FOR OUTPUT AS #1        'Bildschirm
GOSUB Ausgabe
CLOSE #1
OPEN "LPT1:" FOR OUTPUT AS #1        'Drucker LPT1
GOSUB Ausgabe
CLOSE #1
OPEN "TEST.DAT" FOR OUTPUT AS #1     'sequentielle Datei "TEST.DAT"
GOSUB Ausgabe
END

Ausgabe:
  PRINT #1, "Hallo"
RETURN
```

Aufbereitung von Daten für die Druckerausgabe

Um Daten beim Ausdruck aufzubereiten, stehen Ihnen in Power Basic mehrere Möglichkeiten zur Verfügung. Zum einen können Sie über *LPRINT USING* die Ausgabe formatieren, zum anderen können Sie sich Ihre Ausgaben für den Drucker über die Stringfunktionen, die wir bereits kennengelernt haben, selbst aufbauen und formatieren. Wir werden in unserer Druckerausgabe die letzte Möglichkeit wählen. Dabei stehen uns auch die Grafikzeichen aus dem ASCII-Zeichensatz zur Verfügung, mit denen wir bereits unsere Fensterroutine realisiert haben. Bei der Druckerausgabe werden wir diese speziellen Zeichen nutzen, um die Linien für unsere Tabelle zu zeichnen.

Ein Seitenkopf für die Druckerausgabe

Aus der Textverarbeitung kennen Sie vielleicht die Begriffe Kopf- und Fußzeile. Dabei handelt es sich um Zeilen, die jeweils wiederkehrend am oberen bzw. unteren Blattrand erscheinen. Auch Tabellen oder Druckerausgaben sollten über einen Seitenkopf markiert werden, der die Zusammengehörigkeit, aber auch die Reihenfolge des Dokumentes wiedergibt. Wir werden in unserem Seitenkopf, den wir für unser Programm HAUSHALT programmieren wollen, den Programmnamen und die Seitennummer eintragen. Außerdem ist es sinnvoll, den Tabellenkopf in diesen Seitenkopf zu integrieren, damit wir auf jeder Druckerseite eine Spaltendefinition für unsere Tabellenspalten vorfinden.

Verwendung von Drucker-Steuercodes

Über sogenannte Drucker-Steuercodes läßt sich, wie der Ausdruck bereits sagt, der Drucker ansteuern. So können Sie z.B. einen Seitenvorschub, Umstellung von Drucker-Schriftarten, Zeilenabständen, Textauszeichnungen (z.B. kursiv, fett) usw. vornehmen. Die Sache hat allerdings den Haken, daß die Codes von Hersteller zu Hersteller variieren. Ein Programm, das z.B. hervorragende Druckerausgaben auf einem Epson-Drucker macht, muß noch lange nicht für andere Drucker brauchbar sein. Da allerdings ein Großteil der Epson-Druckercodes auch bei anderen Druckern identisch ist und weil viele der heutigen Drucker eine Epson-Emulation anbieten, kann man sich bei Grundbefehlen an diese Druckercodes halten. Um das Programm HAUSHALT allerdings allgemein einsetzbar zu halten, werden wir auf umfangreichere Steuercodes in unserem Programm verzichten.

Die Steuercodes selbst werden, ebenso wie normaler Text, über die *LPRINT*-Anweisung zum Drucker geschickt (alternativ *PRINT#*).

Beispiele:

```
LPRINT CHR$(12);        'Seitenvorschub
LPRINT CHR$(27)+"@";    'Druckerinitialisierung
LPRINT CHR$(27)+"E";    'Fettdruck ein
LPRINT CHR$(27)+"F";    'Fettdruck aus
```

Sämtliche Druckeranweisungen, die Steuercodes enthalten, sollten mit einem Semikolon abgeschlossen werden. Nähere Informationen zu Druckersteuercodes können Sie Ihrem Druckerhandbuch entnehmen.

Praxis

Auch in diesem Kapitel werden wir ausschließlich mit der Datei HAUSHALT.BAS arbeiten. Zunächst müssen wir wieder das Hauptprogramm für unsere Druckerausgabe aufbereiten (Listing 2.6.1). Da die Druckerauswertung nur möglich ist, wenn eine Arbeitsdatei aktiviert und somit geöffnet wurde, verwenden wir erneut die Routine *KeineDateiAktiv*.

```
: 'unveränderte Anweisungen
:
CASE 4     'Ergebnisse ausdrucken
  IF ArbeitsDatei$="keine" THEN
      CALL KeineDateiAktiv(vfarbe%, hfarbe%)
    ELSE
      CALL DatenAuswertung(2, DNr, vfarbe%, hfarbe%)
  END IF
CASE 5     'Grafik
:
: 'unveränderte Anweisungen
```

Listing 2.6.1: An Druckerausgabe angepaßtes Hauptprogramm

Beachten Sie, daß wir, anders als bei der Bildschirmauswertung, die Routine *DatenAuswertung* nun mit dem Wert 2 für den Unterprogrammparameter *Mode%* aufrufen. Dadurch können wir, abhängig vom Parameter Mode%, im Unterprogramm die Druckerausgabe durchführen.

Datenauswertung und Druckerausgabe

Zunächst wollen wir nun unsere Routine *DatenAuswertung* so erweitern, daß über den Parameter *Mode%=2* die Kontrolle der Druckerausgabe durchgeführt wird. Um abhängig von diesem Parameter Druckbefehle abzuschicken oder nicht, bietet sich der Einbau von *IF*-Abfragen an. Die überarbeitete Routine *DatenAuswertung* ist in Listing 2.6.2 abgedruckt. Wesentliche Änderungen sind durch Fettdruck hervorgehoben. Beachten Sie, daß wir zum Teil Blöcke des Quelltextes weglassen, wenn hier keine Änderungen erforderlich sind. Diese Stellen werden durch Kommentarzeilen dokumentiert.

```
SUB DatenAuswertung(Mode%, Dateinummer, vfarbe%, hfarbe%)
  FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
        4 AS Betrag$
  CALL Hintergrund(vfarbe%, hfarbe%)
  CALL Fenster(10, 16, 46, 1, vfarbe%, hfarbe%)
  Jahr$ = RTRIM$(RIGHT$(Date$, 4))
     IF Mode%=1 THEN
         Teil$ = "Bildschirm"
        ELSEIF Mode%=2 THEN
```

```
            Teil$ = "Drucker"
        END IF
    CALL Eingabe("I|"+ Teil$+"auswertung für welches Jahr: ", Jahr$,_
                4, 11, 18, vfarbe%, hfarbe%)
    IF LEN(Jahr$)<> 4 THEN EXIT SUB
    LOCATE ,,0
    FOR SatzNummer&=1 TO LOF(Dateinummer)\50
      GET #Dateinummer, SatzNummer&
      Datum$ = BuchungsDatum$
      Kommentar$ = Beschreibung$
      Betrag! = CVS(Betrag$)
      IF RIGHT$(Datum$, 4) = Jahr$ THEN
        IF Betrag! < 0 THEN                          'Gesamtsummen
            Ausgabe! = Ausgabe! + ABS(Betrag!)
          ELSEIF Betrag! > 0 THEN
            Einnahme! = Einnahme! + Betrag!
        END IF
           IF Mode%=2 THEN                    'Druckerausgabe der Buchungssätze
              Summe! = Einnahme! - Ausgabe!
              CALL DruckerAusgabe(SatzNummer&, Datum$, Kommentar$,_
                   Betrag!, Summe!)
            END IF
        SELECT CASE MID$(Datum$, 4, 2)              'monatlicher Saldo
        :
        : 'unveränderte Inhalt von SELECT CASE
        :
        END SELECT
      END IF
    NEXT SatzNummer&
    Gesamt! = Einnahme! - Ausgabe!
    IF Mode% = 1 THEN  'Bildschirmausgabe wenn Mode%=1
        CALL Fenster(5, 5, 70, 15, vfarbe%, hfarbe%)
        :
        : 'unveränderte Bildschirmausgabe
        :
        CALL Warte
    ELSEIF Mode% = 2 THEN 'Druckerausgabe Gesamtsaldo
           LPRINT : LPRINT
           LPRINT Space$(3)+ "Auswertungsjahr.........:   ";Jahr$
           LPRINT Space$(3)+ "Summe aller Einnahmen...: ";Runden!(Einnahme!)
           LPRINT Space$(3)+ "Summe aller Ausgaben....: ";Runden!(Ausgabe!)
           LPRINT Space$(3)+"Gesamtsaldo............: ";Runden!(Summe!)
           CALL FormFeed
    END IF
    CALL Hintergrund(vfarbe%, hfarbe%)
  END SUB
```

Listing 2.6.2: Erweiterung der Routine DatenAuswertung um die Druckerausgabe

In unserer Routine haben wir drei wesentliche Änderungen vorgenommen. Zunächst haben wir bei der Abfrage, für welches Jahr die Buchungsausgabe vorgenommen werden soll, den Vorgabestring erweitert, damit der

Anwender auch nach der Menüauswahl die Auswertungsart auf dem Bildschirm erkennen kann (Bild 2.6.1).

Bild 2.6.1: Abfrage zur Drucker-auswertung

Diese Erweiterung wurde aus optischen Gründen eingefügt, wäre also nicht unbedingt erforderlich.

```
IF Mode%=2 THEN          'Druckerausgabe der Buchungssätze
  Summe! = Einnahme! - Ausgabe!
  CALL DruckerAusgabe(SatzNummer&, Datum$, Kommentar$, Betrag!, Summe!)
END IF
```

Diese zweite wesentliche Änderung, die innerhalb der *FOR-NEXT*-Schleife steht, und nur abgearbeitet wird, wenn *Mode%=2* ist, sorgt für die Ausgabe der einzelnen Buchungssätze. Da wir nach jedem Buchungssatz einen Zwischensaldo ausgeben wollen, verwenden wir eine weitere mathematische Formel. Die Tabellenwerte werden direkt an die Routine *DruckerAusgabe* übergeben, die für die Tabellenausgabe verantwortlich ist. Dieses Unterprogramm werden wir später noch besprechen.

Die Ausgabe der Gesamtstatistik unserer Auswertung hat nach der Schleife, wenn also alle Ergebnisse feststehen, zu erfolgen. Der Druck der Ergebnisse ist direkt in der Routine *DatenAuswertung* formuliert. Es wäre denkbar, ein neues Unterprogramm hierfür anzulegen. Für die Ausgabe verwenden wir die bereits besprochene *LPRINT*-Anweisung. Damit das letzte Blatt aus dem Drucker ausgeworfen wird, lösen wir durch das Unterprogramm *FormFeed* einen Seitenvorschub aus.

Unterprogramme zur Druckerausgabe

Auch die Unterprogramme zur Druckerausgabe geben wir in das Programm HAUSHALT.BAS am Ende des Quelltextes (nach der Routine *DatenAuswertung*) ein (Listing 2.6.3). Sie werden in diesem Abschnitt kennenlernen, wie eine Vielzahl von Routinen zusammenwirken kann und daß größere Arbeiten in mehrere überschaubare Teilabschnitte zerlegt werden können. Wir werden die einzelnen Teilroutinen nacheinander vorstellen, auch wenn die Entwicklung zum Teil parallel erfolgt ist.

Stellen Sie sich vor, Sie haben Ihre Routine zur Druckerausgabe bereits korrekt erstellt, wollen die Formatierung der Ausgabe aber ändern oder verbessern. Also programmieren Sie sich eine Funktion, die diese Formatierung durchführt und bauen diese anschließend innerhalb Ihrer zuvor entwickelten Routine ein.

☞ Bedenken Sie, daß benutzerdefinierte Routinen in der ersten Fassung nicht unbedingt in der Endfassung vorliegen müssen. Bei der Druckerausgabe könnten Sie z.B. zunächst überprüfen, ob die Ergebnisse richtig übergeben wurden, indem Sie sie unformatiert ausgeben. Im Anschluß daran können Sie die Formatierung vornehmen. Ebenso können Sie bei großen Blockstrukturen vorgehen (*SELECT CASE*, *IF*-Block), indem Sie zunächst die einzelnen Bedingungen formulieren und im Anschluß daran die Anweisungen zu diesen einzelnen Bedingungen eingeben.

Wie Sie in Listing 2.6.3 sehen, wird in der Routine *FormFeed* lediglich das Druckersteuerzeichen für den Seitenvorschub zum Drucker (LPT1) geschickt. Die Routine *SeitenKopf* ist ebenfalls recht einfach zu verstehen, da sie nur über *LPRINT*-Anweisungen Informationen ausdruckt. Sie gibt an das aufrufende Programm die Anzahl der im Seitenkopf gedruckten Zeilen über den Parameter *ZeilenZahl%* zurück. Durch diese Formulierung ist es möglich, den Seitenkopf zu ändern und eine eventuelle neue Zeilenzahl anzugeben. Das Programm arbeitet dann weiterhin korrekt. Wie Sie sehen, formulieren wir den Tabellenkopf direkt mit Grafikzeichen aus dem ASCII-Zeichensatz.

⚠ **Für Grafiksonderzeichen sind keine Tasten auf der Tastatur vorgesehen. Um diese dennoch eingeben zu können, sollten Sie sich das Zeichen aus der ASCII-Tabelle heraussuchen, den Dezimalwert ermitteln und über [Alt] + [Dezimalwert] über die Tastatur eingeben. Wollen Sie z.B. das Sonderzeichen »=« eingeben, so geben Sie dieses über [Alt] + [205] ein. Sie drücken also die [Alt]-Taste und tasten dabei die Zahlenkombination 205 auf dem getrennten numerischen Zehnerblock (am rechten Tastaturende) ein.**

Die aktuelle Seitenzahl, die wir in unserem Seitenkopf ausgeben wollen, erhalten wir über das aufrufende Programm über den Parameter *Seite%*.

```
SUB FormFeed
  LPRINT CHR$(12);
END SUB


SUB SeitenKopf(Seite%, ZeilenZahl%)
  Seite% = Seite% + 1
  LPRINT
  LPRINT SPACE$(3) + "HAUSHALT Version 1.00  - Druckerausgabe Datenauswer-
tung -  Seite: ";Seite%
  LPRINT
  LPRINT
  LPRINT SPACE$(3) + "Nr.   |Buchung   |"+_
        "Kommentar                          |Buchung  |Saldo    "
  LPRINT SPACE$(3) + "-------+-----------+"+_
        "-------------------------------------+---------+----------"
  ZeilenZahl% = 6
END SUB


SUB DruckZeile(Nr&, Datum$, Kommentar$, Betrag!, Summe!)
  Nr$ = SollLen$(LTRIM$(STR$(Nr&)), 6)
  Betrag$ = LTRIM$(STR$(Betrag!))
  Summe$ = LTRIM$(STR$(Summe!))
  LPRINT SPACE$(3) + Nr$ +  "|" + Datum$ +  "|" + Kommentar$ +  "|" +_
        Formatiere$(Betrag$, 9) +  "|" + Formatiere$(Summe$, 10)
END SUB


SUB DruckerAusgabe(SatzNummer&, Datum$, Kommentar$, Betrag!, Summe!)
 STATIC ZeilenZahl%, Seite%
 IF ZeilenZahl% = 0 OR ZeilenZahl% = 60 THEN
   CALL SeitenKopf(Seite%, ZeilenAnzahl%)
 END IF
 CALL DruckZeile(SatzNummer&, Datum$, Kommentar$, Runden!(Betrag!),_
                Runden!(Summe!))
 ZeilenZahl% = ZeilenZahl% + 1
 IF ZeilenZahl% = 60 THEN
   CALL FormFeed
   ZeilenZahl% = 0
 END IF
END SUB

FUNCTION Formatiere$(Zahl$, Laenge%)
  Komma% = INSTR(Zahl$, ".")
  IF Komma% = 0 THEN
      Zahl$ = Zahl$ + ".00"
    ELSEIF INSTR(Zahl$, ".") = LEN(Zahl$)-1 THEN
      Zahl$ = Zahl$ + "0"
  END IF
```

```
  IF Laenge%> LEN(Zahl$) THEN
    Zahl$ = SPACE$(Laenge%-LEN(Zahl$)) + Zahl$
  END IF
  Formatiere$ = Zahl$
END FUNCTION
```

Listing 2.6.3: Routinen für die Druckerausgabe im Programm HAUSHALT

Der Aufbau der Tabellenzeile erfolgt durch das Unterprogramm *DruckZeile*. Sämtliche Werte, die in der Tabelle ausgegeben werden sollen, werden über die Parameterliste an diese Routine übergeben. Eine Erläuterung ist in der nachfolgenden Aufstellung noch einmal gegeben.

```
Nr&        : Datensatznummer bzw. Buchungssatznummer
Datum$     : Buchungsdatum (deutsches Format)
Kommentar$: Kommentar zur Buchung
Betrag!    : Buchungsbetrag
Summe!     : Zwischensaldo
```

Beachten Sie, daß sämtliche numerischen Informationen zunächst mit Hilfe der *STR$*-Funktion in Zeichenketten umgewandelt werden. Da diese Funktion das erste Zeichen bei positiven Zahlenwerten als Leerzeichen darstellt, spalten wir dieses über die *LTRIM$*-Funktion zunächst wieder ab. Auf diesem Wege würde die Zahl 3.56 zur Zeichenkette " *3.56*" und schließlich zu "*3.56*". Der Aufbau der Tabellenzeile erfolgt innerhalb der *LPRINT*-Anweisung, in der auch die benutzerdefinierte Funktion *Formatiere$* aufgerufen wird.

Die Funktion *Formatiere$* sorgt dafür, daß sämtliche in Zeichenketten umgewandelte numerische Werte nach dem Runden genau zwei Nachkommastellen erhalten. Dazu wird die Stelle des Dezimalpunkts ermittelt. Ist noch kein Komma vorhanden, muß es sich um eine Ganzzahl handeln, und wir können direkt die Zeichenfolge "*.00*" an unsere Zeichenkette anhängen. Befindet sich das Komma an der vorletzten Stelle (dann: *Kommaposition=LEN(Zahl$)-1*), so brauchen wir nur eine "*0*" zu ergänzen. Da Zahlen in der Regel rechtsbündig ausgegeben werden, aber von links nach rechts der Ausgabestring aufgebaut wird, muß unsere Zahl auf eine bestimmte Anzahl von Zeichen mit Leerzeichen aufgefüllt werden. Dies erreichen wir durch die Anweisung *Zahl$=SPACE$(Laenge%-LEN(Zahl$))+Zahl$*. Den Parameter für die Länge erhalten wir durch das aufrufende Programm, in unserem Fall durch das Unterprogramm *DruckZeile*.

Die Hauptsteuerung der Druckerausgabe wird durch das Unterprogramm *DruckerAusgabe* durchgeführt. Erinnern Sie sich daran, daß diese Routine innerhalb der Datenauswertung in Abhängigkeit des Parameters *Mode%*

aufgerufen wird. Innerhalb dieses Unterprogramms werden die zuvor erläuterten Unterprogramme verwendet.

Beachten Sie die Anweisung *STATIC* in der Routine *DruckerAusgabe*. Damit werden die Variablen *ZeilenZahl* und *Seite%* als statische Variablen definiert. Zwischen den Aufrufen von *DruckerAusgabe* bleibt der Wert dieser Variablen erhalten. Erinnern Sie sich, daß standardmäßig neu eingeführte Variablen lokale Variablen sind, der Wert würde also zwischen den einzelnen Aufrufen verlorengehen. Zur Veranschaulichung sollten Sie sich die nächsten Beispiele ansehen.

1. Beispiel:

```
SUB Test
  a%=a%+1
  PRINT a%
END SUB

CALL Test  'Ausgabe → 1
CALL Test  'Ausgabe → 1
END
```

2. Beispiel:

```
SUB Test
  STATIC a%
  a%=a%+1
  PRINT a%
END SUB

CALL Test  'Ausgabe → 1
CALL Test  'Ausgabe → 2
END
```

Damit können wir auf die Übernahme von *Zeilenzahl%* und Seite% in die Parameterliste verzichten, da diese Werte ausschließlich in der Routine *DruckerAusgabe*, nicht aber im Hauptprogramm benötigt werden. Wir wollen je Seite 60 Zeilen Text ausgeben. Daher erhöhen wir die Kontrollvariable *ZeilenZahl%* nach jeder Ausgabe einer Druckzeile, führen bei 60 Zeilen einen Seitenvorschub (*CALL FormFeed*) durch und setzen die Variable *ZeilenZahl%* wieder auf 0.

Den Tabellenausdruck, den unsere Druckroutine erzeugt, sehen Sie im folgenden Beispielausdruck (Tabelle 2.6.1). Der Übersichtlichkeit halber haben wir für den Testdruck eine verkleinerte Schrift verwendet. Die Original-Druckerausgabe erfolgt in der voreingestellten Standardschrift des verwendeten Druckers.

Beispielausdruck:

```
HAUSHALT Version 1.00 - Druckerausgabe Datenauswertung -  Seite:  1
Nr.    |Buchung    |Kommentar                          |Buchung  |Saldo
-----------------------------------------------------------------------
1      |11.05.1991|Kommentar 1                         |  -500.00|  -500.00
2      |11.05.1991|Kommentar 2                         |   400.00|  -100.00
3      |11.05.1991|Kommentar 3                         |   100.10|      .10
4      |11.05.1991|Kommentar 4                         |   400.44|   400.54
5      |11.05.1991|Kommentar 5                         |     5.55|   406.09
6      |11.05.1991|Kommentar 6                         |     4.66|   410.75
7      |11.05.1991|Kommentar 7                         |     7.77|   418.52
8      |11.05.1991|Kommentar 8                         |     8.88|   427.40
9      |11.05.1991|Kommentar 9                         |    -9.99|   417.41
10     |11.05.1991|Kommentar 10                        |    10.10|   427.51
11     |11.05.1991|                                    |     0.00|   427.51

Auswertungsjahr.............:  1991
Summe aller Einnahmen.......:  937.5
Summe aller Ausgaben........:  509.99
Gesamtsaldo.................:  427.51
```

Bild 2.6.2: Beispielausdruck der Druckerroutine im Programm HAUSHALT

Workshop

Checkliste

1. Welche Anweisungen zur Druckersteuerung stehen Ihnen in Power Basic zur Verfügung?
2. Welche Möglichkeiten haben Sie, um Tabellen bei der Druckerausgabe zu generieren?
3. Wozu wird ein Seitenkopf verwendet?
4. Was sind Drucker-Steuercodes und wozu werden sie eingesetzt?
5. Erläutern Sie die »statischen Variablen«!

Ideen für eigene Übungen

- Ermöglichen Sie eine Druckerausgabe für beliebige Zeiträume, ohne die Gesamtberechnung und die jeweiligen Berechnungen der Zwischensummen in den einzelnen Zeilen der Tabelle zu beeinflussen!
- Erweitern Sie die Druckerausgabe so, daß auch die monatlichen Gesamtsalden mit ausgegeben werden.
- Formatieren Sie die Druckerausgabe für Gesamteinnahmen, Gesamtausgaben und Gesamtsaldo so, daß die numerischen Werte rechtsbündig formatiert sind und jeweils zwei Nachkommastellen aufweisen!

2.7 Siebte Sitzung: Daten grafisch dargestellt

Die Themen dieser Sitzung:

- Grafikmodus
- Grafische Anweisungen von Power Basic
- Makroprogrammierung mit DRAW
- Koordinatensysteme
- Kreis- und Liniendiagramme

Information

An dieser Stelle werden wir uns mit der Grafikprogrammierung unter Power Basic beschäftigen, wobei es sich um eines der komplexesten Gebiete in der Programmierung handelt. Für unsere Einnahmen-/Ausgabenverwaltung wollen wir grafische Ausgaberoutinen entwickeln. Dabei soll der Verlauf der monatlichen Einzelsalden in einem Liniendiagramm und der Gesamtsaldo in einem Kreisdiagramm veranschaulicht werden. Um die Routinen überschaubar zu halten, werden wir ausschließlich einen bestimmten Grafikmodus benutzen, der später noch genauer zu erläutern ist. Wir wollen die Routinen aber so formulieren, daß eine Anpassung an beliebige Grafikmodi, ohne viel Arbeitsaufwand, möglich ist.

Die Grafikmodi

Sind die in unseren ersten Sitzungen entwickelten Routinen recht unabhängig von der Hardware, so werden wir die Grafikroutinen ausschließlich für eine VGA-Grafikkarte programmieren. Verfügt Ihr Rechner nicht über diese Grafikkarte, müssen Sie eine Anpassung der Routinen vornehmen. Zwar können Sie Grafikprogramme für beliebige Grafikkarten auf jedem Rechner programmieren, auf dem auch Power Basic lauffähig ist, eine Ausführung dieser Programme ist allerdings nicht unbedingt gewährleistet. Die folgende Liste gibt die derzeit auf dem Markt weit verbreiteten

Grafikkarten und die dabei jeweils zur Verfügung stehenden Power-Basic-Grafikmodi wieder (Tabelle 2.7.1).

Grafikkarte	mögliche SCREEN-Modi
CGA	0, 1 , 2
HGC (Hercules)	0, 2, 3
EGA	0, 7, 8, 9, 10
VGA oder MCGA	0, 11, 12

Tabelle 2.7.1: Grafikkarten und mögliche Bildschirmmodi

EGA-, MCGA- und VGA-Karten sind in der Regel abwärtskompatibel und ermöglichen es bei entsprechendem Monitor, auch die CGA- und HGC-Karte zu emulieren. VGA- und MCGA-Karte emulieren zudem zusätzlich die EGA-Modi. Bei älteren Rechnern wurde zum Teil auf den Einbau einer Grafikkarte verzichtet. In diesem Fall können Sie keine Grafiken auf Ihrem Rechner darstellen.

Die Einstellung des Grafikmodus erfolgt innerhalb von Power Basic mit Hilfe des *SCREEN*-Befehls. Je nach gewähltem Bildschirmmodus steht Ihnen eine bestimmte Auflösung zur Verfügung. Im Textmodus entspricht die Auflösung den Textspalten und den Textzeilen, im Grafikmodus spricht man von sogenannten Pixeln. Jeder Pixel besteht aus genau einem Bildschirmpunkt, der durch die Bildschirmkoordinaten und ein eventuelles Farbattribut bestimmt wird. In der Tabelle sind die vom jeweilig vorhandenen *SCREEN*-Modi und die je nach Grafikkarte bedingten Bildschirmauflösungen zusammengestellt.

SCREEN-Modus	Grafikkarte	Farben	Auflösung in Pixeln
1	CGA	4	320x200
2	CGA	2	640x200
	HGC	2	720x348
3	HGC	2	720x348
7	EGA	16	320x200
8	EGA	16	640x200
9	EGA	4/16	640x350
10	EGA	#	640x350 (#=4 Attribute)
11	MCGA/VGA	2	640x480
12	MCGA/VGA	16	640x480

Tabelle 2.7.2: Grafische Bildschirmmodi, Farben und Bildschirmauflösungen

Da wir in unserem Programm HAUSHALT davon ausgehen, daß eine VGA-Grafikkarte vorhanden ist und wir die beste grafische Auflösung einschließlich Farbunterstützung nutzen wollen, werden wir den *SCREEN*-Modus 12 verwenden.

☞ Der VGA-Modus mit 256 Farben und einer Auflösung von 320x200 Pixeln wird derzeit noch nicht von Power Basic unterstützt.

Je nach Grafikkarte steht eine unterschiedliche Anzahl von Bildschirmseiten zur Verfügung. Die Seitenumschaltung erfolgt ebenfalls mit der SCREEN-Anweisung.

Allgemein:

```
SCREEN Modus, Zielseite, Farbflag, AktivSeite
```

In unserem Programm benötigen wir die Seitenumschaltung nicht, deswegen wollen wir auf eine ausführlichere Erläuterung dieser Anweisung an dieser Stelle verzichten. Wollen wir z.B. lediglich in den *SCREEN*-Modus 12 umschalten, genügt der Befehl *SCREEN 12*.

⚠ **Bei der Ausführung einer *SCREEN*-Anweisung wird automatisch in den gewählten Grafikmodus gewechselt und der Bildschirm gelöscht.**

Grafische Anweisungen

Wir haben in Power Basic eine Vielzahl sehr leistungsfähiger Grafikroutinen zur Verfügung, die allerdings nicht mit Grafikbibliotheken konkurrieren können, wie sie z.B. in den modernen Pascal- und C-Dialekten vorhanden sind. Dennoch sind die Anweisungen für die Programmierung von grafischen Standardausgaben, wie wir sie auch in unserem Programm HAUSHALT benötigen, ausreichend. Wir wollen hier nicht sämtliche Anwei-

sungen vorstellen, sondern nur die Befehle anführen, die zum Zeichnen geeignet sind.

Zeichenbefehle mit Kurzerläuterung:	
CIRCLE	Kreis oder Kreisausschnitt zeichnen
DRAW	Makrosprache zum Zeichnen von Punkten, Linien usw.
LINE	Linien, Quadrate und Rechtecke zeichnen
PAINT	Grafikbereiche ausmalen (füllen)
PSET	einzelne Pixel zeichnen
PRESET	einzelne Pixel löschen

Tabelle 2.7.3: Zeichenanweisungen in Power Basic

Man kann bei den Grafikanweisungen zwischen pixel- und vektororientiert unterscheiden. Bei den pixelorientierten Befehlen wird immer genau ein Bildpunkt geschrieben (*PSET*) oder gelöscht (*PRESET*). Mit diesen Befehlen können Sie beliebige grafische Funktionen erzeugen, was insbesondere für die Darstellung von mathematischen Funktionen (z.B. y=sin(x)) notwendig ist. Die Befehle haben allerdings den Nachteil, daß sie verhältnismäßig langsam und somit zum Aufbau komplexer Bildschirmgrafiken nicht geeignet sind. Programmieren Sie z.B. mit Hilfe der *PSET*-Funktion eine Routine zum Zeichnen einer Linie, müssen Sie die Bildschirmkoordinaten für jeden Bildpunkt, der zwischen dem Anfangs- und Endpunkt liegt, selbst berechnen.

Vektororientierte Befehle, wie z.B. *LINE*, benötigen von geometrischen Figuren lediglich einige spezielle Daten und zeichnen das Gebilde anschließend eigenständig auf den Bildschirm. Die Berechnung von Zwischenwerten erfolgt automatisch, z.B. bei einer Linie, von einem angegebenen Anfangspunkt zu einem bestimmten Endpunkt.

Die meisten Zeichenprogramme, die auf dem Markt erhältlich sind, arbeiten pixelorientiert, d.h. Grafiken werden punktweise gespeichert. Programme zum Erstellen technischer Zeichnungen (CAD) hingegen arbeiten vektororientiert. Eine Linie wird dabei nicht mehr punktweise gespeichert, sondern lediglich durch den Anfangs- und Endpunkt.

Makroprogrammierung mit DRAW

Eine spezielle Anweisung steht mit *DRAW* zur Verfügung. Dabei handelt es sich um eine einfache Makrosprache zum Zeichnen von Grafiken. *DRAW* erkennt eine Vielzahl von Kommandos, die eine Art Turtle- bzw. Schildkrötengrafik ermöglichen. Ausgehend von einer beliebigen Position

kann ein Zeichencursor in eine beliebige Richtung bewegt werden. Eine Drehung um einen bestimmten Winkel ist dabei ebenso möglich, wie das Füllen von Flächen oder das einfache Setzen des Zeichencursors. Die Zusammenstellung der *DRAW*-Kommandos können Sie Tabelle 2.7.4 entnehmen. *DRAW* erwartet die Kommandos in String- bzw. Zeichenkettenformat. Durch die Anweisung *DRAW "U20"* könnten Sie z.B. den Zeichencursor um 20 Bildpunkte nach oben bewegen lassen. Aber auch Variablen können in den Kommandostring eingebaut werden.

Beispiel:

```
SCREEN 12
Z%=40 : R$="U20 R30 U20 D40"
DRAW "D="+VARPTR$(Z%)
DRAW "BM 200, 200 X"+ VARPTR$(R$)
```

Handelt es sich um eine variable Zeichenkette, verwenden Sie, wie in dem vorangegangenen Beispiel dargestellt, ein »X« und bei einer Zahl ein »=« zur Kennzeichnung. Power Basic verwendet eine andere Stringverwaltung als zum Beispiel der Vorgänger Turbo Basic. Damit Sie Routinen, die *DRAW* verwenden, auch in späteren Versionen von Power Basic nutzen können, sollten Sie immer mit der *VARPTR$*-Funktion arbeiten. Wir verwenden in unserer Routine eine andere Möglichkeit, indem wir numerische Werte über die *STR$*-Funktion in eine Zeichenkette umwandeln. In diesem Fall ist allerdings nicht gewährleistet, daß die Routine auch später noch unter Power Basic lauffähig ist.

Kommando	Beschreibung
Un	Zeichencursor um n Pixel nach oben
Dn	Zeichencursor um n Pixel nach unten
Ln	Zeichencursor um n Pixel nach links
Rn	Zeichencursor um n Pixel nach rechts
En	Zeichencursor um n Pixel nach oben rechts
Fn	Zeichencursor um n Pixel nach unten rechts
Gn	Zeichencursor um n Pixel nach unten links
Hn	Zeichencursor um n Pixel nach oben links
Mx,y	Zeichencursor positionieren
B	Zusatzbefehl: Cursor nur bewegen, nicht zeichnen
N	Zusatzbefehl: nach Zeichenbefehl an ursprüngliche Position
Aw	Winkel w festlegen
TAw	Drehung um den Winkel w
Cf	Zeichenfarbe f festlegen
Ss	Skalierungsfaktor auf s/4 setzen
Pf,g	Bereich mit Farbe f füllen (g Grenzfarbe)

Tabelle 2.7.4: DRAW-Kommandos

Koordinatensysteme

Vielleicht kennen Sie aus der Mathematik noch die Koordinatensysteme. Zweidimensionale Funktionen werden z.B. in ein X/Y-Koordinatensystem eingetragen. Auch die Routine zum Zeichnen der monatlichen Zwischensalden muß in ein zweidimensionales Koordinatensystem gelegt werden.

Die erste Möglichkeit, die wir haben, ist, daß wir die bekannte Bildschirmauflösung für unser Koordinatensystem berücksichtigen (physikalisches Koordinatensystem). Zum anderen könnten wir uns unser eigenes Koordinatensystem mit dem *WINDOW*-Befehl festlegen (globales Koordinatensystem). In diesem Fall werden die globalen Koordinaten in unserem Programm automatisch in die physikalischen Koordinaten umgerechnet. Der Einsatz der Anweisung *WINDOW* ist vor allen Dingen dann sinnvoll, wenn die darzustellenden Werte größer als die Bildschirmauflösung in Pixeln sind. Ist dies der Fall, und Sie verwenden *WINDOW* nicht, müssen Sie selbst für die korrekte Umrechnung sorgen. Wir werden in unserem Programm HAUSHALT auf den *WINDOW*-Befehl verzichten und mit dem festen physikalischen Koordinatensystem arbeiten.

Kreis- und Liniendiagramme

Zur Veranschaulichung von Zahlenwerten verwendet man die sogenannte Geschäftsgrafik. Neben den Kreis- bzw. Kuchendiagrammen und den Balken- bzw. Stabdiagrammen, sind die Liniendiagramme die am häufigsten verwendeten Grafikformen. Läßt sich in Kreisdiagrammen in der Regel nur eine begrenzte Anzahl von Werten veranschaulichen, können mit den Linien- und Balkendiagrammen theoretisch beliebig viele Werte dargestellt werden. Für unser Programm zur Einnahmen-/Ausgabenverwaltung wollen wir das Kreisdiagramm für den Gesamtsaldo und das Liniendiagramm für die monatlichen Zwischensalden verwenden. Das Kreisdiagramm ist dabei sehr einfach über den Befehl *CIRCLE* und das Liniendiagramm über den Befehl *LINE* realisierbar.

Praxis

Mit der Grafikausgabe wollen wir in dieser Sitzung die eigentlichen Anweisungen für unser Programm HAUSHALT abschließen. In den nächsten Sitzungen werden wir uns dann mit der Programmoptimierung befassen.

Für die grafische Auswertung müssen wir zunächst das Hauptprogramm und anschließend noch einmal unsere Routine *DatenAuswertung* anpassen, in der der Aufruf für die grafischen Ausgaben erfolgen soll. Im Anschluß daran werden wir eine Routine zum Zeichnen eines Kreisdiagramms und eine Routine zum Zeichnen eines Liniendiagramms entwickeln.

Wir werden sämtliche Änderungen und Programmerweiterungen in der Datei HAUSHALT.BAS vornehmen, die sich immer noch im Editor befindet. Zunächst sollten Sie mit Hilfe der Cursorsteuerung das Hauptprogramm ansteuern und die nötigen Änderungen in der *SELECT-CASE*-Auswahl, wie in Listing 2.7.1, vornehmen.

```
:
: 'unveränderte Anweisungen des Hauptprogramms
:
CASE 5    'Grafik
  IF ArbeitsDatei$="keine" THEN
      CALL KeineDateiAktiv(vfarbe%, hfarbe%)
    ELSE
      CALL DatenAuswertung(3, DNr, vfarbe%, hfarbe%)
  END IF
CASE 6    'Tools (OS-Shell)
:
: 'unveränderte Anweisungen im Hauptprogramm
:
```

Listing 2.7.1: Anpassung des Hauptprogramms an die Grafikausgabe

Auch für die grafische Datenauswertung müssen wir zunächst wieder über die *IF*-Abfrage kontrollieren, ob bereits eine Arbeitsdatei ausgewählt und geöffnet wurde. Ist dies nicht der Fall, wird keine Grafikausgabe durchgeführt und lediglich eine Meldung für den Programmanwender ausgegeben (*CALL KeineDateiAktiv*). Um der Routine *DatenAuswertung* zu zeigen, daß Grafiken erstellt werden sollen, übergeben wir für *Mode%* nun einen Wert ungleich 1 (Bildschirmausgabe) und ungleich 2 (Druckerausgabe). In diesem Fall haben wir den Wert 3 gewählt. Sie werden später anhand des Quelltextes im Listing 2.7.2 sehen, daß auch andere Werte zum selben Ergebnis geführt hätten.

Datenauswertung und Grafikausgabe

Nachdem wir die *SELECT-CASE*-Anweisung im Hauptprogramm erweitert haben, müssen wir nun zunächst die Änderungen an der Routine *DatenAuswertung* vornehmen (Listing 2.7.2). Sämtliche wesentlichen Änderungen sind zur Veranschaulichung wieder durch Fettdruck hervorgehoben.

```
SUB DatenAuswertung(Mode%, Dateinummer, vfarbe%, hfarbe%)
  :
  : 'unveränderte Anweisungen
  :
  IF Mode%=1 THEN
      Teil$ = "Bildschirm"
    ELSEIF Mode%=2 THEN
      Teil$ = "Drucker"
    ELSE
      Teil$ = "Grafik"
  END IF
  CALL Eingabe("I|"+ Teil$+"auswertung für welches Jahr: ", Jahr$,_
                4, 11, 18, vfarbe%, hfarbe%)
  : 'unveränderte Anweisungen
  IF Mode%=1 THEN
      : 'unveränderte Bildschirmausgabe
    ELSEIF Mode%=2 THEN
      : 'unveränderte Druckerausgabe
    ELSEIF Mode% = 3 THEN              'Kreisdiagramm
        SCREEN 12                       'Grafikmodus
        CALL Linien(Januar!,Februar!, Maerz!, April!, Mai!, Juni!, Juli!,_
             August!, September!, Oktober!, November!, Dezember!)
        CLS
        IF Einnahme!<>0 AND Ausgabe!<>0 THEN
          CALL Kreis(Einnahme!, Ausgabe!)
        END IF
        SCREEN 0                        'zurück in Textmodus
  END IF
  CALL Hintergrund(vfarbe%, hfarbe%)
END SUB
```

Listing 2.7.2: Grafikerweiterung in der Routine DatenAuswertung

Die erste Erweiterung in der Routine wurde aus rein optischen Gründen eingefügt. Wird nun eine Grafikauswertung durchgeführt, so wird das für die Berechnung zu berücksichtigende Buchungsjahr abgefragt, wie in Bild 2.7.1 dargestellt.

Die wesentlichen Anweisungen für die Grafikerstellung sind am Ende der Routine formuliert. Dann also, wenn sämtliche Ergebnisse, die für das Kreis- und Liniendiagramm benötigt werden, bereits berechnet sind.

Beachten Sie, daß schon in der Routine *DatenAuswertung* mit der *SCREEN*-Anweisung in den Grafikmodus umgeschaltet wird. Grafische Ausgaben ohne Umschaltung in den Grafikmodus führen zu einem Laufzeitfehler.

Bild 2.7.1: Abfrage des Buchungsjahres für die Grafikauswertung im Programm HAUSHALT

Als erste grafische Ausgabe wird ein Liniendiagramm über das Unterprogramm *Linien* gezeichnet. Die zwölf Werte für die monatlichen Zwischensalden werden über die Parameterliste übergeben. Farbwerte und Bildschirmbegrenzungen sind fest vorgegeben und werden später im Unterprogramm selbst initialisiert. Ist die Routine *Linien* für alle Werte lauffähig, und wenn keine kritischen Berechnungen im Unterprogramm erfolgen, wird vor dem Aufruf des Unterprogramms *Kreis* zunächst überprüft, ob Werte für die Einnahmen und Ausgaben ungleich Null vorhanden sind. Erst dann ist eine sinnvolle Darstellung in einem Kreisdiagramm möglich. Die zwei Werte, die an *Kreis* übergeben werden, müssen in dem Unterprogramm in Werte umgerechnet werden, die eine Darstellung in einem entsprechenden Kreissegment bzw. Kuchenstück ermöglichen. Zwischen den beiden grafischen Darstellungen wird der Bildschirm über die *CLS*-Anweisung gelöscht. Erinnern Sie sich daran, daß auch die *SCREEN*-Anweisung den Bildschirm löscht, und zwar in unserem Fall vor der ersten und nach der letzten grafischen Ausgabe. Über die Anweisung *SCREEN 0* wird vom Grafik- in den Textmodus zurückgeschaltet.

Das Kreisdiagramm

Nachdem wir das Hauptprogramm und die Routine *DatenAuswertung* an die grafischen Ausgaben angepaßt haben, können wir uns nun der Grafikprogrammierung widmen. Sowohl die Anweisungen zum Kreisdiagramm als auch die Anweisungen für das Liniendiagramm sollen in eigenen Unterprogrammen formuliert werden. Diese geben wir in den Editor wieder am Ende des Quelltextes ein, dort, wo auch die übrigen benutzerdefinierten Routinen vorhanden sind.

Der Grafikmodus, den wir verwenden, hat eine Auflösung von 640 x 480 Pixel. Unser Unterprogramm *Kreis* soll ein Kreisdiagramm zeichnen, in dem lediglich zwei Werte gegenübergestellt werden. Die Einnahmesumme soll dabei der Gesamtheit des Kreises, also 100 % bzw. 360 entsprechen. Die Ausgabesumme hingegen soll als Teilstück des Kreises berücksichtigt werden. Dabei gehen wir davon aus, daß die Ausgaben unter der Summe der Einnahmen liegen, der Gesamtsaldo also positiv ist. Bezogen auf Altgrad ergeben sich dann die folgenden mathematischen Beziehungen:

```
Einnahmen = 100% bzw. 360
Einnahmen/360= 1% bzw. 1
Teilkreis = Ausgaben / (Einnahmen/360)
```

Mit der letztgenannten Formel ist es uns möglich, umzurechnen, wie groß der Anteil der Ausgaben an den Einnahmen ist. Diese Formel haben wir auch in der Routine *Kreis* verwendet. Dazu ist allerdings zu berücksichtigen, daß in Power Basic sämtliche Kreisfunktionen und auch die *CIRCLE*-Anweisung spezielle Werte in Bogenmaß und nicht in Altgrad erwarten. Da wir unsere Überlegungen aber in Altgrad gemacht haben, müssen wir eine Funktion entwickeln, die eine Umrechnung von Altgrad in Bogenmaß vornimmt. Die Formel für die Umrechnung kann jeder Formelsammlung entnommen werden. Es gilt:

```
360 = 2 * PI
PI  = 3.141592654
```

Die Funktion *Grad2Rad#* und das Unterprogramm *Kreis* sind in Listing 2.7.3 abgedruckt.

```
SUB Kreis(Einnahme!, Ausgabe!)
  maxX% = 640-1                          'Grafikmodus
  maxY% = 480-1
  G! = Ausgabe!/(Einnahme!/360)
  LOCATE 30,1                            'Text / Untergrund
  COLOR 1, 3
  PRINT "Fortsetzung mit beliebiger Taste...";
  COLOR 14                               'Beschriftung
  LOCATE 3, 20
  PRINT "Kreisdiagramm Einnahmen/Ausgaben Gesamt in DM"
  LOCATE 4, 20
  PRINT "Gesamteinnahmen: ";Runden!(Einnahme!);"DM"
  COLOR 4
  LOCATE 16, 61
  PRINT "Saldo    =";Runden!(Einnahme!-Ausgabe!)
  COLOR 5
  LOCATE 15, 61
```

```
  PRINT "Ausgaben=";Runden!(Ausgabe!)
  LINE (0, 0) - (maxX%, maxY%), 15, B      'Rahmen
  LINE (0, maxY%-16) - (maxX%, maxY%-16), 15
  CIRCLE (maxX%\2, maxY%\2), 150, 15       'Kreisdiagramm
  PAINT (maxX%\2, maxY%\2), 5, 15
  CIRCLE (maxX%\2, maxY%\2), 150, 15, -0.001, (-1) * Grad2Rad#(G!)
  PAINT (maxX%\2+10, maxY%\2+2), 2,15
  CALL Warte                               'warten auf Tastendruck
END SUB

FUNCTION Grad2Rad#(G!)
  pi# = 3.141592654
  Grad2Rad# = ((2 * pi#)/360) * G!
END FUNCTION
```

Listing 2.7.3: Routinen zum Zeichnen eines Kreisdiagramms im VGA-Modus

Nun wollen wir die Funktion ein wenig genauer betrachten. Zunächst wird PI – wie oben bereits erwähnt – initialisiert. Danach wird der übergebene Wert *G!* in Altgrad in Bogenmaß umgerechnet. Dabei entspricht ein Altgrad *(2*PI)/360*. Demnach ist die Umrechnung in der Funktion *Grad2Rad#* korrekt erfolgt.

⚠ **Grafikprogrammierung ist immer sehr eng mit der Programmierung von mathematischen Funktionen verbunden. Grundkenntnisse der Mathematik sind daher bei der grafischen Programmierung unbedingt erforderlich.**

In der Routine *Kreis* werden zunächst die Bildschirmbereiche initialisiert (640 x 480 Pixel). Die Bildbreite geht dabei von Pixel 0 bis 639 und die Bildtiefe von Pixel 0 bis 479. Anschließend wird, wie bereits erläutert, der Anteil der Ausgaben an den Einnahmen berechnet. Danach folgt ein allgemeiner Bildschirmaufbau mit Textinformationen (*LOCATE-PRINT*-Anweisungen) und das Zeichnen eines Bildschirmrahmens mit Hilfe der *LINE*-Anweisung. Beachten Sie, daß der Text entsprechend der VGA-Auflösung positioniert wurde, und bei anderen Grafikmodi unter Umständen Korrekturen erforderlich sind. Die Grafikbefehle hingegen sind so formuliert, daß die Änderung der Variablen *maxX%* und *maxY%* am Beginn des Unterprogramms die wichtigsten Anpassungen an eine andere Bildschirmauflösung vornehmen. Mit Hilfe der *CIRCLE*-Anweisungen wird dann das Kreisdiagramm gezeichnet.

Vereinfachte *CIRCLE*-Anweisung:

```
CIRCLE (x,y),Radius,Farbe,Anfang,Ende
```

Die *CIRCLE*-Anweisung benötigt zunächst Koordinaten für den Kreismittelpunkt (x und y), den Radius im gewählten Koordinatensystem und eine Zeichenfarbe.

Wird nur ein Kuchenstück gezeichnet, müssen zusätzlich Werte für den Anfangs- und Endwinkel in Bogenmaß gemacht werden. Diese Werte sind in unserem Programm negativ, damit der Anfang und das Ende des Bogens mit dem Kreismittelpunkt verbunden werden, also ein »Kuchenstück« entsteht. Mit der *PAINT*-Anweisung können wir die geschlossenen Linien der jeweiligen Kreisfläche mit einer Farbe füllen lassen. Dazu muß der Zeichencursor sich innerhalb des Kreises bzw. des entstandenen Kuchenstücks befinden. Dies erreichen wir dadurch, daß wir für *PAINT* den Cursor minimal in Richtung Füllfläche bewegen (Listing). Die Funktion zum Berechnen eines Wertes von Altgrad in Bogenmaß wird innerhalb der *CIRCLE*-Anweisung aufgerufen.

Damit die Programmausführung unterbrochen wird, die Grafik also zunächst auf dem Bildschirm stehen bleibt, rufen wir die in der ersten Sitzung entwickelte Routine *Warte* auf. Das Programm wird dann nur fortgesetzt, wenn eine beliebige Taste gedrückt wird.

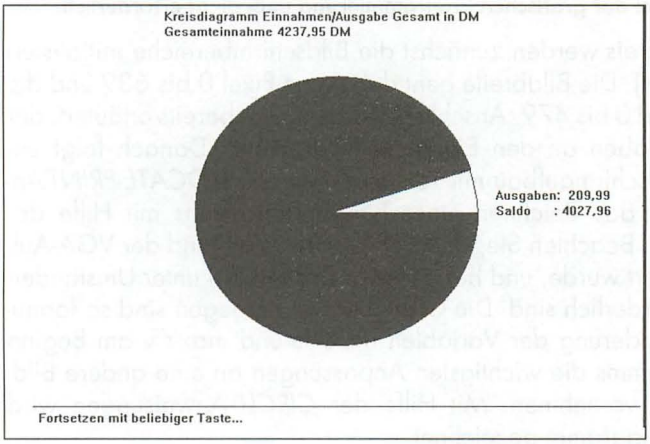

Bild 2.7.2: Bildschirmausgabe eines Kreisdiagramms im Programm HAUSHALT

Die Bildschirmausgabe, die sich durch die Routine *Kreis* ergibt, ist in Bild 2.7.2 dargestellt.

Das Liniendiagramm

Nun können wir uns unserer zweiten Grafik zuwenden. Auch hier werden zunächst die Werte für die Bildschirmauflösung definiert. Im Anschluß daran wird ein X/Y-Koordinatensystem mit *LINE* und eine passende Beschriftung mit den Befehlen *LOCATE* und *PRINT* auf den Bildschirm gezeichnet.

⚠ **Eine Beschriftung von Grafiken ist in Power Basic nur über die *PRINT*-Anweisung möglich. Dieser Befehl hat allerdings den Nachteil, daß Text nicht pixelorientiert, sondern wie im Textmodus nur zeilen- und spaltenweise ausgegeben werden kann. Über die Makrosprache *DRAW* wäre es möglich, sich vorher genau definierte Buchstaben auch pixelorientiert ausgeben zu lassen.**

Um Pfeilspitzen an unserem Koordinatensystem zu realisieren, verwenden wir den Makrobefehl *DRAW*. Erst danach kann der Linienzug der monatlichen Gesamtsalden in unser Koordinatensystem eingetragen werden.

Dazu verwenden wir eine *FOR-NEXT*-Schleife, die durch Anwendung einer *SELECT-CASE*-Anweisung sämtliche Monatswerte in chronologischer Reihenfolge auswertet. Durch die letzte *DRAW*-Anweisung vor dieser Schleife wird der Zeichencursor zunächst auf den Koordinatenursprung gesetzt. In der Schleife selbst wird dann von Wert zu Wert eine Linie mit *LINE* gezeichnet (Listing 2.7.4). Aus diesem Grunde brauchen wir nie einen Anfangs-, sondern immer nur den Endpunkt der Linie anzugeben.

```
SUB Linien(Jan!,Feb!,Mar!,Apr!,Mai!,Jun!,Jul!,Aug!,Sep!,Okt!,Nov!,Dez!)
  maxX% = 640-1                          'Grafikmodus
  maxY% = 480-1
  LOCATE 30,1                            'Text / Untergrund
  COLOR 1, 3
  PRINT "Fortsetzung mit beliebiger Taste...";
  COLOR 14                               'Beschriftung
  LOCATE 3, 25
  PRINT "Salden bezogen auf einzelne Monate"
  LOCATE 4, 20
  LINE (0, 0) - (maxX%, maxY%), 15, B    'Rahmen
  LINE (0, maxY%-16) - (maxX%, maxY%-16), 15
  LINE (10, maxY%\2) - (maxX%-10, maxY%\2), 1
  DRAW "C1 H10 F10 G10"                  'Pfeil rechts
  LINE (20, maxY%-30) - (20, 35), 1
  DRAW "C1 G10 E10 F10"                  'Pfeil rechts
  DRAW "BM 10,"+STR$(maxY%\2)            'Ursprung
  FOR x%=1 TO 12                         'x-Koordinatenleiste
    DRAW "R45 D5 U5"
  NEXT x%
  COLOR 1
  LOCATE  2, 3: PRINT "DM*1000"
```

```
  LOCATE 16, 2: PRINT "0"
  LOCATE 16,72: PRINT "Monat"
  LOCATE 17, 6: PRINT "Jan"
  LOCATE 17,24: PRINT "Apr"
  LOCATE 17,46: PRINT "Aug"
  LOCATE 17,68: PRINT "Dez"
  DRAW "BM 20,"+STR$(maxY%\2)              'y-Koordinatenleiste
  FOR x%=1 TO 3
    DRAW "D60 R5 L5"
  NEXT x%
  DRAW "BM 20,"+STR$(maxY%\2)              'y-Koordinatenleiste
  FOR x%=1 TO 3
    DRAW "U60 R5 L5"
  NEXT x%
  LOCATE  4,5: PRINT "+15"
  LOCATE  8,5: PRINT "+10"
  LOCATE 12,5: PRINT "+5"
  LOCATE 19,5: PRINT "-5"
  LOCATE 23,5: PRINT "-10"
  LOCATE 27,5: PRINT "-15"                 'Liniengrafik
  DRAW "BM 55,"+STR$((maxY%\2)-Pixelwert%(Jan!))
  xWert%=55
  FOR x% = 2 TO 12
    SELECT CASE x%
      CASE  2: Monat! = Feb!
      CASE  3: Monat! = Mar!
      CASE  4: Monat! = Apr!
      CASE  5: Monat! = Mai!
      CASE  6: Monat! = Jun!
      CASE  7: Monat! = Jul!
      CASE  8: Monat! = Aug!
      CASE  9: Monat! = Sep!
      CASE 10: Monat! = Okt!
      CASE 11: Monat! = Nov!
      CASE 12: Monat! = Dez!
      CASE ELSE
    END SELECT
    xWert% = xWert%+45
    LINE -(xWert%,(maxY%\2)-PixelWert%(Monat!)), 14
  NEXT x%
  CALL Warte
END SUB

FUNCTION PixelWert%(Wert!)
  PixelWert% = INT(Wert! * (60/5000))
END FUNCTION
```

Listing 2.7.4: Routinen zum Zeichnen eines Liniendiagramms im VGA-Modus

Da wir in dieser Routine ein physikalisches Koordinatensystem verwenden, müssen wir die Umrechnung der Werte, die in dieses Koordinatensystem einzutragen sind, selber vornehmen. Dies erfolgt über die Funktion *PixelWert%*. Dazu sind zunächst wieder einige mathematische Vorüber-

legungen notwendig. Eine Einheit auf unserer Y-Achse soll 5000 DM betragen. Sie soll ferner 60 Pixel in unserer Grafik entsprechen. Demnach gilt für einen bestimmten Wert:

```
Pixel = Wert * (60/5000)
```

Diese Formel verwenden wir auch in der Routine *PixelWert%* und konvertieren das erhaltene Ergebnis in eine Ganzzahl (Datentyp Integer), da nur ganze Pixelwerte in unserer Grafik berücksichtigt werden können. Die Funktion wird innerhalb der *LINE*-Anweisung aufgerufen, die unsere Ergebnisse auf den Bildschirm zeichnet.

Nachdem die Grafik komplett auf dem Bildschirm gezeichnet ist, wird die Programmausführung über die Anweisung *CALL Warte* wieder so lange unterbrochen, bis der Anwender das Programm durch eine beliebige Taste fortsetzt. Eine Ergebnisgrafik der Routine *Linien* ist in Bild 2.7.3 dargestellt.

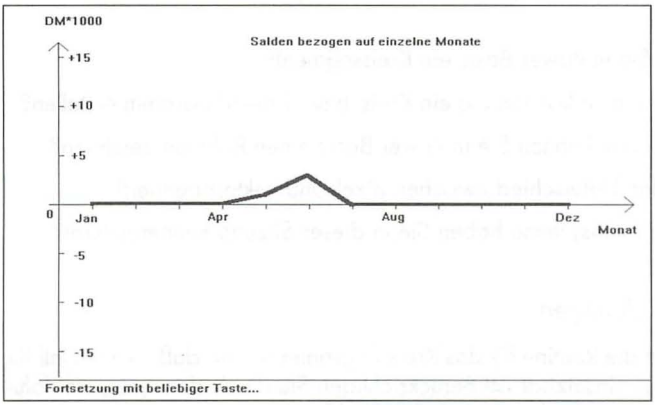

Bild 2.7.3: Bildschirmausgabe eines Liniendiagramms im Programm HAUSHALT

Für allgemeinere Formulierungen eines Liniendiagrammes sollten Sie das Zeichnen des Koordinatensystems und das Zeichnen einzelner Linien in getrennten Unterprogrammen formulieren. Dann ist es auch sehr leicht möglich, mehrere Linien in ein Diagramm zu zeichnen, ohne daß das Koordinatensystem jedesmal neu gezeichnet wird und unter Umständen bereits gezeichnete Linien überschreibt.

Nach der Fertigstellung ist unser Programm zur Einnahmen-/Ausgabenüberschußberechnung eigentlich schon fertig. Wir können es nun mit der Tastenfolge [Alt] + [F9] und [F9] in Maschinensprache übersetzen und ausführen lassen. Sicherlich werden Sie bemerken, daß noch nicht alle

Menüeinträge in unserem Programm HAUSHALT belegt sind, und unter Umständen Fehlermeldungen das Programm vorzeitig beenden.

Ein Fehler tritt z.B. auf, wenn Sie die Druckerausgabe im Programm starten, der Drucker aber nicht bereit ist. Aus diesem Grunde werden wir nun mit der Optimierungsphase unserer Programmentwicklung beginnen.

Workshop

Checkliste

1. Was bedeutet der Grafikmodus für die Programmierung?
2. Wie kann zwischen den unterschiedlichen Grafikmodi in Power Basic umgeschaltet werden?
3. Welche Möglichkeiten bietet Ihnen die DRAW-Anweisung?
4. Nennen Sie die wichtigsten grafischen Anweisungen von Power Basic und Ihre Möglichkeiten!
5. Wie zeichnen Sie in Power Basic ein Kreissegment?
6. Mit welchen Befehlen können Sie ein Kreis- bzw. Liniendiagramm erstellen?
7. Mit welchem Befehl können Sie in Power Basic einen Rahmen zeichnen?
8. Erläutern Sie den Unterschied zwischen pixel- und vektororientiert!
9. Welche Koordinatensysteme haben Sie in dieser Sitzung kennengelernt?

Ideen für eigene Übungen

- Formulieren Sie die Routine für das Kreisdiagramm so um, daß sie speziell für den EGA-Modus einsetzbar ist! Berücksichtigen Sie die Änderungen, die dafür im Hauptprogramm erforderlich sind (*SCREEN*-Befehl)!
- Wie wir bereits erwähnt haben, bietet sich die Verwaltung der monatlichen Zwischensalden in einem Datenfeld an. In diesem Fall wäre der Prozedurkopf der Routine *Linien* z.B. in *SUB Linien(Salden!())* umzuändern. Nehmen Sie die weiteren Anpassungen in diesem Unterprogramm für den Aufruf mit dem Datenfeld vor und formulieren Sie den Unterprogrammaufruf, wenn Ihr Datenfeld im aufrufenden Programm den Namen *MonatSalden!()* hat.
- Ersetzen Sie die *STR$*-Funktion in der Routine *Linien* durch die *VARPTR$*-Funktion, damit die Routine auch für spätere Power-Basic-Versionen einsetzbar ist!

2.8 Achte Sitzung: Optimierung durch Fehlerbehandlung

Die Themen dieser Sitzung:

- Syntaxfehler und Laufzeitfehler
- Abfangen von Laufzeitfehlern
- Möglichkeiten der Programmierung mit der *ON-ERROR*-Anweisung

Information

In dieser Sitzung wollen wir unser Programm HAUSHALT mit einer Fehlerbehandlungsroutine ausstatten. Fehler, die während der Programmausführung auftreten, sollen nicht mehr zum Programmabbruch führen. Statt dessen soll für den Anwender eine aussagekräftige Fehlermeldung auf dem Bildschirm ausgegeben werden. Diese soll den Fehler – wenn möglich – abstellen. Im Anschluß, nach der Fehlermeldung, soll das Programm fortgeführt werden.

Syntaxfehler und Laufzeitfehler

In der Programmierung unterscheidet man zwischen den Syntax- oder Kompilier-Fehlern und den sogenannten Laufzeitfehlern. Werden die Syntaxfehler bereits bei der Übersetzung durch den Compiler erkannt, ergeben sich die Laufzeitfehler erst während der Programmausführung.

Geben Sie z.B in einem Programm die Zeile *PRIN "Hallo Freunde!"* ein, meldet der Compiler bereits während der Umsetzung in Maschinensprache den Fehler *Error 415: "=" expected*, und der Textcursor des Editors springt an den Anfang der entsprechenden Zeile. Natürlich sehen Sie sofort, daß es sich hier um einen Schreibfehler des Schlüsselwortes *PRINT* handelt, der sich umgehend korrigieren läßt. Damit wäre der Syntaxfehler beseitigt. Die Fehlermeldung bedarf an dieser Stelle noch einer kurzen Erläuterung. Der Compiler muß zunächst überprüfen, ob es sich in der jeweiligen Zeile um Befehle handelt, die in Power Basic definiert sind, wie z.B. der Befehl *PRINT*, oder ob es sich um benutzerdefinierte

Befehle oder Variablen handelt. In unserem Fall nimmt der Compiler an, daß es sich bei *PRIN* um eine Variable handelt. Dieser Variablen wird ein konstanter Wert, nämlich eine Zeichenkette, zugewiesen. Die Zuweisung erfolgt immer durch das »=«-Zeichen, das allerdings in unserer Zeile fehlen würde. Dies erklärt zunächst die Fehlermeldung des Compilers.

Sie sehen, daß die Fehlermeldung nicht immer eine korrekte Lösung bietet. Bei den Fehlern, die durch den Compiler aufgespürt werden, handelt es sich aber um recht leicht korrigierbare Fehler. In der Regel sind lediglich Schreibfehler zu verbessern.

Alle Fehler, die nicht vom Compiler entdeckt werden, sind sogenannte Laufzeitfehler: Fehler, die erst bei der Programmausführung und häufig nur bei bestimmten Sonderfällen auftreten. Durch spezielle Programmiertechniken kann eine Vielzahl von möglichen Laufzeitfehlern bereits bei der Programmerstellung berücksichtigt werden. Man kann grob zwei Arten von Laufzeitfehlern unterscheiden. Zunächst gibt es die Laufzeitfehler, die durch den Quelltext selbst hervorgerufen werden (z.B. Überschreiten von Bereichsgrenzen eines definierten Datenfeldes). Diese können zum Teil durch spezielle Programmsteuerungen unterdrückt werden. Weniger Möglichkeiten hätte man bei den Laufzeitfehlern, die sich durch das Rechnersystem selbst ergeben (z.B. Drucken bei ausgeschaltetem Drucker). In diesem Fall tritt also ein Laufzeitfehler auf, obwohl der Quelltext korrekt ist. Für solche Fälle stellt Power Basic mit der *ON-ERROR*-Anweisung eine komfortable Möglichkeit zur Fehlerbehandlung bereit, die wir nachfolgend noch genauer besprechen werden.

Abfangen von Laufzeitfehlern

Wie bereits mehrfach in unserem Programm HAUSHALT geschehen, kann das Auftreten von Fehlern während der Laufzeitphase durch sogenannte Steuerungsvariablen vermieden werden. So ist z.B. keine Datenauswertung möglich, solange keine Arbeitsdatei gewählt wurde. Man muß natürlich berücksichtigen, daß sämtliche Anweisungen zur Kontrolle von Programmvariablen, wie es auch in unserem Fall über die Variable *Arbeitsdatei$* geschieht, zusätzlichen Programmcode erzeugt. Das heißt, unser Programm wird um so größer, je mehr Kontrollen und eventuelle Korrekturen wir einbauen. Nehmen wir hierzu ein Beispiel. Ein mathematisches Ergebnis soll durch folgende Anweisung errechnet werden:

```
Ergebnis! = 1000 / Variable%
```

Würden wir für *Variable%* nun den Wert Null einsetzen, ergäbe sich eine unzulässige Division durch Null. Diesen Fall könnten wir recht schnell über eine *IF*-Abfrage ausschließen.

```
IF Variable% <> 0 THEN
  Ergebnis! = 1000 / Variable%
END IF
```

So weit, so gut. Nun müssen wir allerdings weiterhin bedenken, daß unser Ergebnis eventuell im weiteren Programmverlauf für weitere Berechnungen benötigt wird. Wurde kein entsprechender Wert für *Ergebnis!* initialisiert, wäre dieser Wert gleich Null und würde aller Voraussicht nach zu weiteren Fehlern führen. Unser erster Versuch, den Fehler abzufangen, kann also auch ganz das Gegenteil bewirken. Es sieht dann zwar so aus, als ob das Programm korrekt arbeiten würde, die Endergebnisse wären allerdings falsch. Das Aufspüren dieser Fehler ist äußerst schwierig. Daher überdenken Sie zunächst, welche Konsequenz eine programmiertechnische Korrektur haben kann. Auch ist es nicht unbedingt erforderlich, sämtliche Variablen vor der Auswertung auf Korrektheit zu überprüfen. Hierzu wollen wir ein kleines Beispiel anführen.

```
'Hauptprogramm
Zeile%=30
Spalte%=1
CALL Meldung (Zeile%,Spalte%)
CALL Meldung (25, 7)
END

'Unterprogramm
SUB Meldung (Zeile%,Spalte%)
  LOCATE Zeile%, Spalte%
  PRINT "Dies ist eine Meldung";
END SUB
```

Für die Variable *Zeile%* sind im Normalfall nur die Werte 1 bis 25 und für die Variable *Spalte%* nur die Werte 1 bis 80 zulässig. Da die Festlegung von Bildschirmausgaben in der Regel im Hauptprogramm bereits fest erfolgt, lohnt es sich natürlich nicht, sämtliche Übergabeparameter des Unterprogramms zu überprüfen. Falsche Angaben führen dann bereits während der Testphase zu Fehlermeldungen im Entwicklungssystem von Power Basic.

Natürlich können nicht sämtliche Fehler auf diesem Wege abgefangen werden. Mit der Fehlerverfolgung über die *ON-ERROR*-Anweisung haben wir allerdings die Möglichkeit, auch die unwahrscheinlichsten Fehler zu berücksichtigen. Benutzen Sie *ON-ERROR* nicht, und es tritt ein Laufzeitfehler in einem ausführbaren Maschinenprogramm auf, wird das Pro-

gramm unmittelbar beendet und auf Kommandozeilenebene zurückgekehrt. Auf dem Bildschirm erscheint dann eine Fehlermeldung in der nachfolgenden Form.

```
Error 75 at pgm-ctr: 1334
```

Die Möglichkeiten, die Sie haben, um einen so gemeldeten Laufzeitfehler in Power Basic zu lokalisieren, werden wir in unserem Kapitel zur Fehlerbeseitigung noch besprechen.

Die ON-ERROR-Anweisung

Mit der *ON-ERROR*-Anweisung haben Sie die Möglichkeit, eine Fehlerbehandlungsroutine festzulegen, die beim Auftreten eines Laufzeitfehlers angesprungen werden soll. Die Fehlerverfolgung können Sie jederzeit aktivieren und wieder ausschalten.

Anweisungen zur Fehlerverfolgung:

```
Sprungmarke definieren: ON ERROR GOTO label
Verfolgung abschalten : ON ERROR GOTO 0
```

Beachten Sie, daß, wenn die Fehlerverfolgung in einem ausführbaren Programm wieder abgeschaltet ist, unter Umständen das Programm wieder mit einer Fehlermeldung abgebrochen wird. Der Beginn der Fehlerbehandlungsroutine wird durch ein sogenanntes Label bzw. eine Sprungmarke eingeleitet und immer mit einer *RESUME*-Anweisung beendet.

RESUME-Anweisung:

```
RESUME oder : Programm mit der Anweisung fortsetzen in
RESUME 0    : der der Fehler aufgetreten ist

Resume NEXT : Programm mit der Anweisung fortsetzen, die
              nach der Anweisung folgt, in der der Fehler
              aufgetreten ist

RESUME Label: Programm an einer definierten Sprungmarke
              fortsetzen
```

Die *RESUME*-Anweisung darf ausschließlich in Verbindung mit der *ON-ERROR*-Anweisung eingesetzt werden.

In der Tabelle 2.8.1 sind die wichtigsten Laufzeitfehlermeldungen von Power Basic aufgeführt. Fehler 51 ist ein besonders kritischer Fehler. Er besagt, daß ein Fehler innerhalb von Power Basic selbst aufgetreten ist. Eine Beseitigung von seiten des Programmierers ist in diesem Falle nicht möglich. Sollte eines Ihrer Programme diesen Fehler tatsächlich hervor-

rufen, sollten Sie sich direkt an Kirschbaum Software, den deutschen Programmvertreiber, wenden.

Laufzeitfehler	Fehlerbeschreibung
6	Datentyp-Überlauf in einer Berechnung
7	unzureichender Speicher
11	Division durch Null
14	Speicher für Zeichenketten voll
15	Eine Zeichenkette ist zu lang
24	COM-Timeout
25	Peripherieprobleme
27	kein Papier im Drucker
51	interner Power-Basic-Fehler
53	Datei nicht gefunden
55	Datei ist bereits geöffnet
57	Geräte-Ein-/Ausgabefehler
58	Datei ist bereits vorhanden
61	Diskette/Festplatte voll
71	Laufwerksklappe geöffnet
72	Diskette defekt!
75	Datei gegen Zugriff gesperrt
76	Suchpfad ist nicht vorhanden
242	Zeichenkettenspeicher zerstört

Tabelle 2.8.1: Die wichtigsten Laufzeitfehlermeldungen von Power Basic

Der Grundaufbau der Fehlerbehandlung kann an einem einfachen Beispiel veranschaulicht werden.

```
'Hauptprogramm
ON ERROR GOTO Behandlung
:
: 'Programmanweisungen Hauptprogramm
:
END

'Sprungmarke Fehlerbehandlung
Behandlung:
  PRINT "Fehler im Programm!"
RESUME NEXT
```

Die Fehlermeldungen, die für den Anwender ausgegeben werden sollen, können mehr oder weniger umfangreich sein. In der Regel reicht es aus, die nichtssagenden Fehlernummern durch kurze Erläuterungen zu ersetzen. Dies werden wir auch in unserem Programm HAUSHALT tun. In diesem Fall belasten wir den Speicher nicht übermäßig mit Zeichenketten-

konstanten. Sollen die Informationen umfangreicher sein, ist es sinnvoll, die Fehlermeldungen z.B. in einer sequentiellen Datei im ASCII-Format abzulegen. Bei Bedarf kann dann die entsprechende Fehlermeldung eingelesen werden. Viele kommerzielle Anwendungsprogramme lagern die Fehlermeldungen auf diese Art und Weise aus. Die Datei mit den Fehlermeldungen könnte in unserem Programm z.B. den Namen HAUSHALT.ERR erhalten und hätte in etwa folgenden Aufbau:

```
0005  unzulässiger Funktionsaufruf
0007  Arbeitsspeicher nicht ausreichend
0011  Division durch Null
0025  Peripherieprobleme
:
usw.
```

Diese sequentielle Datei können Sie nun zeilenweise lesen. Ist die linke Zahl der Zeile, also *VAL(LEFT$(GeleseneZeile$,4))* identisch mit der aufgetretenen Fehlernummer, so ist die Fehlermeldung ermittelt. Die Fehlerdatei kann an dieser Stelle geschlossen und die Meldung entsprechend ausgegeben werden.

Praxis

Nachdem wir uns entschlossen haben, eine Fehlerbehandlung in unser Programm HAUSHALT zu integrieren und die Grundlagen nun bekannt sind, können wir uns wieder der Programmierung zuwenden. Dazu müssen wir unser Hauptprogramm ändern. Spezielle Unterprogramme und Funktionen werden wir in dieser Sitzung nicht entwickeln.

Das Hauptprogramm und die Fehlerbehandlung

Unsere Fehlermeldung soll zwar knapp, aber auch aussagekräftig sein. Sie soll jeweils in der untersten Bildschirmzeile über die Routine *InfoZeile*, die wir in unsere Funktionsbibliothek integriert haben, ausgegeben werden. Um die Meldung hervorzuheben, wollen wir spezielle Farben für die Ausgabe verwenden. Der Text soll in gelber Schrift auf rotem Hintergrund ausgegeben werden. Dazu definieren wir am Programmanfang zunächst die Variablen *vErr%* und *hErr%* (Listing 2.8.1). Da unsere Fehlerbehandlung für das ganze Programm gelten soll, ist nur eine einmalige *ON-ERROR*-Anweisung am Programmbeginn erforderlich. Die Sprungmarke erhält dabei den Namen *FehlerBehandlung*.

```
$INCLUDE "TOOLS.H"              'Deklarationen der Unit TOOLS
$LINK "TOOLS.PBU"               'Benutzerbibliothek TOOLS einbinden

'*********************************************************************
'* Programmname              : HAUSHALT.BAS Version 1.00             *
'* Compiler                  : Power Basic 2.10                      *
'* verwendete Headerdateien: TOOLS.H                                 *
'* verwendete Units          : TOOLS.PBU                             *
'* geschrieben               : April/Mai 1991                        *
'* letzte Änderung           : 16.05.1991                            *
'*********************************************************************

$STACK 4000                     'Stapelspeicher erhöhen
CLS                             'Hauptprogramm HAUSHALT

ON ERROR GOTO FehlerBehandlung

vfarbe% = 7                     'Daten initialisieren
hfarbe% = 1
vErr%   = 14
hErr%   = 4
Anzahl% = 8                     'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
Menue$(3) = "Datenauswertung"
Menue$(4) = "Druck"
Menue$(5) = "Grafik"
Menue$(6) = "OS-Shell"
Menue$(7) = "Hilfe"
Menue$(8) = "Ende"

Start:
ArbeitsDatei$="keine"
CALL Hintergrund(vfarbe%, hfarbe%)
WHILE Wahl% <> 8
  : 'unveränderte Anweisungen des
  : 'Hauptprogramms
WEND
CLOSE #DNr
COLOR 7, 0
CLS
PRINT "Programm beendet..."
END

'*********************************************************************
'* Fehlerbehandlung und Ereignisverfolgung                           *
'*********************************************************************

Fehlerbehandlung:
  SELECT CASE ERR
    CASE 6
       Fehler$="Überlauf!"
    CASE 14, 201
       Fehler$="unzureichender Speicherplatz!"
    CASE 24, 25, 68, 69
```

```
      Fehler$="Peripherie-, Schnittstellenprobleme!"
   CASE 27, 57
      Fehler$="Der Drucker ist nicht bereit!"
   CASE 51
      Fehler$="Interner Fehler!"
   CASE 70, 71, 72
      Fehler$="Probleme mit dem Laufwerk!"
   CASE 71
      Fehler$="Keine Diskette/Klappe offen!"
   CASE ELSE
      Fehler$="Fehler Nummer "+STR$(ERR)+" ist aufgetreten!"
  END SELECT
  CALL InfoZeile("Achtung: "+Fehler$, 25, vErr%, hErr%)
  CALL Warte
RESUME Start
```

Listing 2.8.1: Die Fehlerbehandlung zum Programm HAUSHALT

Nach diesen Änderungen setzen wir die Sprungmarke *Start* an das Ende des Initialisierungsteiles des Hauptprogramms. An dieser Stelle soll später die Programmausführung fortgesetzt werden, wenn ein Fehler während der Programmausführung auftritt. Die eigentlichen Anweisungen zur Fehlerbehandlung schreiben wir an das Ende des Hauptprogramms und vor den Teil des Quelltextes, in dem die Unterprogramme und Funktionen definiert sind. Wie Sie ja wissen, befinden wir uns immer noch in der Datei HAUSHALT.BAS. Um den Fehlerteil hervorzuheben, leiten wir ihn durch Kommentarzeilen ein. Tritt ein Fehler auf, wird automatisch die Sprungmarke *FehlerBehandlung* angesprungen und die *SELECT-CASE*-Anweisung ausgeführt. Die Power-Basic-Funktion *ERR* liefert uns die Fehlernummer des zuletzt aufgetretenen Fehlers zurück. Abhängig von dieser Nummer wird dann der Variablen *Fehler$* die Zeichenkette mit dem Fehlertext zugewiesen und über die Routine *InfoZeile* ausgegeben. Die Fehlermeldung bleibt so lange in der untersten Bildschirmzeile stehen, bis das Programm mit einer beliebigen Taste fortgesetzt wird (*CALL Warte*). Die *RESUME*-Anweisung ist dann für den Neustart des Hauptprogramms verantwortlich (Rücksprung zum Label *Start*).

⚠ **Beachten Sie, daß das Hauptprogramm nun mit einer *END*-Anweisung abschließen muß, da sonst beim Programmende die Anweisungen nach dem Label *Fehlerbehandlung* auch ohne Laufzeitfehler abgearbeitet werden. Beim Abarbeiten des *RESUME*-Fehlers würde das Programm einen Fehler auslösen (*RESUME ohne ERROR*). Sie sollten jedes Hauptprogramm mit dem *END*-Befehl beenden, auch wenn dies in Power Basic nicht unbedingt erforderlich ist.**

Die Fehlerroutine können Sie sehr schnell testen, indem Sie das Programm mit [Alt] + [F9] [F9] ausführen und anschließend die Druckerausgabe aus dem Hauptmenü des Programms HAUSHALT anwählen. Der Drucker sollte

dabei ausgeschaltet sein. In der untersten Zeile erscheint dann die Fehlermeldung, wie in Bild 2.8.1 dargestellt.

Bild 2.8.1: Fehlermeldung bei nicht bereitem Drucker im Programm HAUSHALT

Workshop

Checkliste

1. Was ist der Unterschied zwischen Laufzeit- und Syntaxfehlern?
2. Auf welche Art und Weise lassen sich Laufzeitfehler abfangen?
3. Erläutern Sie den Aufbau der *ON-ERROR*-Anweisung!

Ideen für eigene Übungen

- Ändern Sie die Routine der Fehlerbehandlung so ab, daß die jeweilige benötigte Fehlermeldung aus einer sequentiellen Datei gelesen wird!
- Überlegen Sie sich eine Möglichkeit, um die Fehlermeldungen in einer Direktzugriffsdatei zu speichern!
- Ändern Sie die Fehlerbehandlung so ab, daß die *SELECT-CASE*-Auswahl nicht mehr nach der Sprungmarke *Fehlerbehandlung*, sondern in einem getrennten Unterprogramm vorhanden ist! Berücksichtigen Sie dabei die Übergabeparameter, die Sie dann für die Routine *InfoZeile* benötigen!

2.9 Neunte Sitzung: Ein Hilfesystem gefällig?

Die Themen dieser Sitzung:

- Sequentielle Dateiverwaltung
- Die *EOF*-Funktion
- Realisierung eines kontext-sensitiven Hilfesystems
- Möglichkeiten der Ereignisverfolgung
- Der *SHELL*-Befehl

Information

In dieser Sitzung wollen wir die Programmierarbeit an unserem Projekt HAUSHALT beenden. Das Programm soll mit einer Hilfsfunktion versehen werden. Außerdem soll eine Schnittstelle zum Betriebssystem eingebaut werden.

Sequentielle Dateiverwaltung

Haben wir für unsere Dateiverwaltung in unserem Programm HAUSHALT mit Direktzugriffsdateien gearbeitet, werden wir das Hilfesystem mit Hilfe der sequentiellen Dateien realisieren. Während Direktzugriffsdateien den wahlfreien Zugriff auf bestimmte Datensätze erlauben, müssen wir sequentielle Daten in der Regel immer vom Dateibeginn bis zum Dateiende lesen. Außerdem können die Datensätze in einer sequentiellen Datei unterschiedliche Längen haben, was bei Randomdateien nicht der Fall ist. Wir wählen für unser Programm diesen Datentyp, da wir die Dateien nur lesen wollen und die Erstellung mit Hilfe des Power-Basic-Editors durchgeführt werden soll. Damit liegt die Datei im reinen ASCII-Format vor und kann direkt im sequentiellen Dateimodus von Power Basic verarbeitet werden.

Im Gegensatz zu Direktzugriffsdateien gibt es für sequentielle Dateien keine einheitliche *OPEN*-Anweisung. Je nachdem, ob eine sequentielle Datei neu erzeugt, verlängert oder nur gelesen wird, sind unterschiedliche Anweisungen im Quelltext zu wählen.

OPEN-Anweisungen für sequentiellen Dateimodus:

```
OPEN Name FOR Modus AS Nummer
```

```
Name   : Dateiname
Nummer: Dateinummer
Modus : INPUT      - lesen
        OUTPUT     - schreiben
        APPEND     - schreiben
```

Power Basic kennt einen sequentiellen Lesemodus und zwei Schreibmodi. Verwenden Sie das Schlüsselwort *OUTPUT* in Ihrer *OPEN*-Anweisung, wird die sequentielle Datei neu erzeugt. Das heißt, eine bereits existierende Datei mit demselben Namen wird zuvor gelöscht. Mit *APPEND* haben Sie die Möglichkeit, eine Datei zu verlängern. Ein eventuell bestehender Dateiinhalt bleibt erhalten, neue Daten werden jeweils an das Dateiende angefügt. Für das Programm HAUSHALT werden wir lediglich den Lesemodus verwenden.

Um in einer sequentiellen Datei zu lesen, bzw. um sequentielle Daten in eine sequentielle Datei zu schreiben, stehen spezielle Anweisungen zur Verfügung. Diese Anweisungen sind äquivalent zu den bildschirmorientierten Anweisungen und benötigen lediglich die jeweilige Dateinummer der geöffneten sequentiellen Datei.

Beispiele:

```
sequentiell lesen     : INPUT #DNr, a%, b%
                        INPUT #DNr, Text$
                        LINE INPUT #DNr, Zeile$
sequentiell schreiben: PRINT #DNr, "Hallo"
                        WRITE #DNr, 3, 7
```

Wir werden uns in unserem Programm auf die Verwendung der *LINE-INPUT#*-Anweisung beschränken, die jeweils komplette Datensätze einschließlich Kommata einliest, was bei der *INPUT#*-Anweisung nicht der Fall ist. Diese interpretiert Kommata als Feldtrennzeichen.

Erstellen Sie eine sequentielle Datei und wollen diese später wieder lesen, so müssen die Schreib- und Lesebefehle dem folgenden Beispiel entsprechen.

```
OPEN "TEST.DAT" FOR OUTPUT AS #1
  a%=5
  b!=3.7
  PRINT #1, a%, b!
CLOSE #1
```

```
OPEN "TEST.DAT" FOR INPUT AS #1
  INPUT #1, c%, d!
  PRINT c%, d!
CLOSE #1
```

Mit diesen Grundlagen werden wir dazu in der Lage sein, ein komfortables Hilfesystem mit sehr wenig Programmieraufwand zu erzeugen.

Die EOF-Funktion

Ist es uns bei den Direktzugriffsdateien möglich, mit Hilfe der *LOF*-Funktion das Dateiende zu ermitteln, steht uns bei den sequentiellen Dateien die *EOF*-Funktion zur Verfügung. Sie erkennt, ob die End-Of-File-Marke, die das Ende einer sequentiellen Datei markiert, erreicht ist. Um zum Beispiel eine Datei bis zur *EOF*-Marke zu lesen, bietet sich die Verwendung dieser Funktion in einer *WHILE-WEND*-Schleife an.

Beispiel:

```
DNr = FREEFILE 'freie Dateinummer ermitteln
OPEN "TEST.DAT" FOR INPUT AS #DNr
WHILE NOT EOF(DNr)
  LINE INPUT #DNr, Zeile$   'aus Datei lesen
  PRINT Zeile$              'auf Bildschirm ausgeben
WEND
CLOSE #DNr
```

Realisierung eines Hilfesystems

Wenn wir davon ausgehen, daß eine Hilfsdatei mit speziellen Informationen bereits vorliegt, stellt sich die Frage, wie und vor allen Dingen wann wir diese Informationen auf dem Bildschirm abrufen lassen wollen. Die einfachste Möglichkeit besteht darin, eine Hilfsfunktion in unser Auswahlmenü zu integrieren und von dort einen Hilfe-Index auszugeben. Aus diesem Index können wir dann ein entsprechendes Thema wählen, zu dem wir nähere Informationen benötigen. Diesen Weg wollen wir auch in unserem Programm HAUSHALT gehen.

Damit haben wir aber auch schon die Grundlage für ein kontextsensitives Hilfesystem gelegt. Jede Hilfsinformation steht in einem engen Zusammenhang mit einem speziellen Schlüsselwort. Theoretisch könnten wir über jedes Schlüsselwort direkt auf eine spezielle Hilfe zugreifen. Kontextsensitiv meint dabei, daß, je nachdem, wann die Hilfe angefordert wird, eine unterschiedliche Information erscheint. Befinde ich mich z.B. bei der Dateneingabe, so bekomme ich Informationen zur Dateneingabe und möglichen Tastaturkommandos. Lasse ich hingegen eine grafische Aus-

wertung vornehmen, erhalte ich nähere Informationen zur Grafik selbst. Kommerzielle Programme stellen in der Regel beide Möglichkeiten zur Verfügung.

Power Basic bietet sowohl eine kontextsensitive Menühilfe als auch eine indexorientierte Hilfsfunktion für den Sprachumfang und die Tastaturkommandos an. Die erste Hilfeform steht zur Verfügung, wenn man sich im Menü befindet, die zweite, wenn Quelltexte bearbeitet werden.

Wir werden die Hilfe für das Programm HAUSHALT auf kurze Informationen für die einzelnen Menüeinträge beschränken.

Die Ereignisverfolgung als Alternative

Um die kontextsensitive Hilfe zu programmieren, können Sie die Ereignisverfolgung von Power Basic nutzen. Über *ON KEY(1) GOSUB Hilfe* würden Sie zum Beispiel beim Drücken der [F1]-Taste automatisch an eine Sprungmarke *Hilfe* gelangen, die die Hilfsinformation aufbereitet. Um die Funktionstaste ein- bzw. auszuschalten, verwenden Sie die Befehle *KEY(1) ON* bzw. *KEY(1) OFF*. Power Basic erzeugt den Code zur Tastaturüberwachung automatisch.

☞ Neben speziellen Tasten können auch weitere Ereignisse in Power Basic verfolgt werden, wie z.B. der Dateneingang der seriellen Schnittstelle. Genauere Informationen erhalten Sie im Referenz-Handbuch unter den Schlüsselwörtern, die in der Tabelle 2.9.1 aufgeführt sind.

Ereignis	Erläuterung
ON KEY	Tastaturverfolgung
ON PEN	Statusänderung eines Lichtgriffels verfolgen
ON PLAY	Musikpufferstatus verfolgen
ON STRIG	Joystick verfolgen
ON TIMER	zeitabhängige Befehlsausführung

Tabelle 2.9.1: Ereignisverfolgung in Power Basic

Damit die Information jeweils zur aktuellen Programmausführung paßt, müssen Sie eine Steuerungsvariable *Hilfe$* einführen. Dieser Hilfsvariablen könnte in diesem Fall das Schlüsselwort zur jeweils passenden Hilfsinformation zugewiesen werden. Befinden Sie sich in der Grafikroutine, ergibt sich z.B. die Zuweisung *Hilfe$="Grafik"*; befinden Sie sich bei der Dateneingabe, so ergibt sich die Zuweisung *Hilfe$="Datenein-*

gabe". Diese Zuweisung erspart die Auswahl aus dem Index, da der Hilfsbegriff bereits festgelegt wurde.

Eine Schnittstelle zum Betriebssystem

Mit dem *SHELL*-Befehl stellt Power Basic eine Anweisung zur Verfügung, um ein Anwendungsprogramm temporär zu verlassen und auf Betriebssystemebene zurückzukehren. Dabei verbleibt das Programm selbst im Arbeitsspeicher. Dieser Befehl kann nur ausgeführt werden, wenn der Arbeitsspeicher genügend groß ist, um eine Kopie des Kommandoprozessors COMMAND.COM nachzuladen. Um später wieder in das Anwendungsprogramm zurückzukehren, braucht nach dem Systemprompt lediglich der Befehl *EXIT* eingetastet zu werden.

Betriebssystemebene:

```
C:\>EXIT [Return]
```

Wird *SHELL* ohne Parameter aufgerufen, gelangt man direkt zur Betriebssystemebene, wie oben beschrieben. Man hat allerdings auch die Möglichkeit, an diesen Befehl direkt einen Kommandostring zu übergeben, der unmittelbar ausgeführt werden soll. Stellen Sie sich vor, Sie wollen die Datei TEST.DAT über den *SHELL*-Befehl löschen. Dieses wäre mit der Anweisung *SHELL "DEL TEST.DAT"* möglich. Der Rückschritt zum Betriebssystem erfolgt in diesem Fall nur zeitweilig, d.h., nach der Befehlsausführung befinden Sie sich automatisch wieder in Ihrem Anwendungsprogramm.

```
zur Rückkehr zum Programm bitte EXIT eintasten...

DR DOS Release 5.0
Copyright (c) 1976,1982,1988 Digital Research Inc. Alle Rechte vorbehalten.
Licensed to VOBIS Data Computer GmbH,Aachen "HIGHSCREEN(TM)".

C:\WORD>EXIT
```

Bild 2.9.1: Die Betriebssystemebene

☞ Beachten Sie, daß diverse Befehle, die über *SHELL* aufgerufen werden, eigenständig Bildschirmausgaben vornehmen, die unter Umständen Ihre

eigene Bildschirmoberfläche zerstören. Um dies zu umgehen, können Sie das DOS-Piping einsetzen, das wir unter dem Kapitel »Kniffe, Tips und andere Spezialitäten« vorstellen werden.

Praxis

Bevor wir die letzten Änderungen in unserem Hauptprogramm vornehmen, wollen wir in dieser Sitzung zunächst eine Hilfsdatei erstellen. Anschließend werden wir die Hilfsfunktion und die Schnittstelle zum Betriebssystem in unser Hauptprogramm integrieren.

Eine Hilfsdatei entsteht

Um eine neue Datei in Power Basic zu erzeugen, wählen Sie über [F10] *File → New*. Anschließend geben wir den Hilfstext ein (Listing 2.9.1). Sie sehen, daß jedes Schlüsselwort in eckigen Klammern eingefaßt und mit Großbuchstaben geschrieben ist. Jede Hilfsinformation zu einem bestimmten Begriff wird durch ein Doppelkreuz abgeschlossen. Damit können wir später eine Interpretation der sequentiellen Datei realisieren. Da die Ausgaben der Hilfe später in einem Bildschirmfenster erscheinen sollen, ist darauf zu achten, daß jede Zeile höchstens 76 Zeichen lang sein darf, um mit einem Rahmen versehen werden zu können.

Inhalt der Hilfsdatei HAUSHALT.HLP:

```
[MENUE]
Menü-Hilfe:
Bevor Sie die einzelnen Menüeinträge anwählen
können, müssen Sie zunächst eine Arbeitsdatei
auswählen. Der Name der derzeit aktiven
Arbeitsdatei wird in der unteren Zeile angezeigt.
#
[DATEIWAHL]
Dateiwahl-Hilfe:
Ist noch keine Arbeitsdatei angelegt, wird automatisch
die Datei HAUSHALT.DAT angelegt. Ansonsten können Sie
auch eine neue Datei erzeugen. Das Programm kann in
dieser Version maximal 10 Dateien verwalten.
#
[DATENEINGABE]
Eingabe-Hilfe:
Sie können zu jeder Buchung einen Kommentar und eine
positive oder negative Buchungssumme eingeben. Als
Buchungsdatum wird das interne Systemdatum gesetzt.
Spezielle Datierungen sind im Kommentarfeld mit unter-
zubringen. Je Datei können maximal 500 000 Buchungen
vorgenommen werden. In einer Datei können mehrere
Buchungsjahre erfaßt werden.
```

```
#
[AUSWERTUNG]
Auswertung:
Die Auswertung der Datentypen erfolgt mit einfacher
Genauigkeit und bezieht sich jeweils auf ein bestimmtes
einzelnes Buchungsjahr. Die jeweiligen Endergebnisse werden
auf zwei Nachkommastellen gerundet. Mit dem Menüpunkt DRUCK
können die einzelnen Buchungen und die Endergebnisse in
Tabellenform ausgegeben werden. Eine grafische Darstellung
der Endergebnisse ist mit dem Menüpunkt GRAFIK möglich.
#
[DRUCKER]
Drucker-Hilfe:
Bei der Druckerausgabe werden keine speziellen Steuercodes
verwendet. Eine Ausgabe ist auf jedem Matrix- und Laserdrucker
möglich. Tritt ein Fehler auf (z.B. Papierstau), so wird die
Ausgabe beendet und unmittelbar in das Hauptmenü zurückgekehrt.
#
[GRAFIK]
Grafik-Hilfe:
Die grafische Ausgabe besteht aus zwei Teilen. Zunächst werden
die monatlichen Salden in einem Liniendiagramm aufgetragen,
um die Gewinne und Verluste zwischen den Monaten eines Jahres
sehr schnell und anschaulich vergleichen zu können. In einem
Kreisdiagramm werden dann die Ausgabensumme und die Gewinnsumme
für den gesamten Buchungszeitraum dargestellt.
```

Listing 2.9.1: Die ASCII-Datei HAUSHALT.HLP

Nachdem der Text eingegeben worden ist, müssen wir ihn in einer neuen Datei speichern. Um die Verbindung zu unserem Programm HAUSHALT zu verdeutlichen und zu zeigen, daß es sich um eine Hilfsdatei handelt (Dateikürzel .HLP), wählen wir den Namen HAUSHALT.HLP. Diesen Namen geben wir ein, nachdem wir mit [F10] *File → Write to* das entsprechende Eingabefenster von Power Basic geöffnet haben. Unsere Eingabe quittieren wir mit [Return]. Im Anschluß daran kehren wir über [Alt] + [F3] in das Programm HAUSHALT.BAS zurück.

Das Hauptprogramm wird vollendet

Nun können wir die letzten Änderungen an unserem Programm HAUSHALT vornehmen. Zunächst fügen wir die Anweisungen für den *SHELL*-Befehl ein, den wir über den sechsten Menüpunkt aktivieren wollen. Dazu schalten wir zunächst in einen Monochrommodus (weiße Schrift auf schwarzem Hintergrund) und löschen den Textbildschirm mit unserer Oberfläche. Danach schreiben wir die Information auf den Bildschirm, wie in das Programm HAUSHALT zurückgekehrt werden kann. Bedenken Sie, daß für den Anwender später nicht mehr erkennbar ist, ob sich das

Programm noch im Speicher befindet oder nicht. Um den Einbau einer späteren Ereignisverfolgung zu erleichtern, werden wir die Hilfsfunktion nach einem Sprunglabel in unser Programm eingeben. Wir fügen es unmittelbar nach der Fehlerbehandlungsroutine in unseren Quelltext ein (Listing 2.9.2). Das bedeutet, daß wir dieses Label über einen *GOSUB*-Befehl anspringen (*GOSUB Hilfe*). Nachdem wir die *SELECT-CASE*-Auswahl abschließend bearbeitet haben, können wir nun die Hilfsroutine erstellen.

```
$INCLUDE "TOOLS.H"              'Deklarationen der Unit TOOLS
$LINK "TOOLS.PBU"               'Benutzerbibliothek TOOLS einbinden

'*********************************************************************
'* Programmname              : HAUSHALT.BAS Version 1.00             *
'* Compiler                  : Power Basic 2.10                      *
'* verwendete Headerdateien: TOOLS.H                                 *
'* verwendete Units          : TOOLS.PBU                             *
'* geschrieben               : April/Mai 1991                        *
'* letzte Änderung           : 16.05.1991                            *
'*********************************************************************

$STACK 4000                     'Stapelspeicher erhöhen
CLS                             'Hauptprogramm HAUSHALT

ON ERROR GOTO FehlerBehandlung

vfarbe% = 7                     'Daten initialisieren
hfarbe% = 1
vErr%   = 14
hErr%   = 4
Anzahl% = 8                     'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
Menue$(3) = "Datenauswertung"
Menue$(4) = "Druck"
Menue$(5) = "Grafik"
Menue$(6) = "OS-Shell"
Menue$(7) = "Hilfe"
Menue$(8) = "Ende"

Start:
ArbeitsDatei$="keine"
CALL Hintergrund(vfarbe%, hfarbe%)
WHILE Wahl% <> 8
  CALL InfoZeile("HAUSHALT Version 1.00 - Einnahmen-"+_
                 "/Ausgabenüberschußrechnung", 1, vfarbe%, hfarbe%)
```

```
  CALL InfoZeile(CHR$(24)+CHR$(25)+": Anwahl Menüeintrag   RETURN:"+_
                 " Menüauswahl    Arbeitsdatei: "+ ArbeitsDatei$ +_
                 Space$(12), 25, vfarbe%, hfarbe%)
  CALL Auswahl(Menue$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
  SELECT CASE Wahl%
    CASE 1   'Arbeitsdatei wählen und öffnen
      IF ArbeitsDatei$ <> "keine" THEN
        CLOSE #DNr
      END IF
      CALL DateiNamen(Namen$(), vfarbe%, hfarbe%, Datei%)
      ArbeitsDatei$ = Namen$(Datei%)
      DNr = FREEFILE
      OPEN ArbeitsDatei$ FOR RANDOM AS #DNr LEN=50
    CASE 2   'Dateneingabe
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenEingabe(DNr, vfarbe%, hfarbe%)
      END IF
    CASE 3   'Datenauswertung
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenAuswertung(1, DNr, vfarbe%, hfarbe%)
      END IF
    CASE 4    'Ergebnisse ausdrucken
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenAuswertung(2, DNr, vfarbe%, hfarbe%)
      END IF
    CASE 5    'Grafik
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenAuswertung(3, DNr, vfarbe%, hfarbe%)
      END IF
       CASE 6    'Tools (OS-Shell)
         COLOR 7,0
         CLS
         PRINT "zur Rückkehr zum Programm bitte EXIT eintasten..."
         SHELL
         CALL Hintergrund(vfarbe%, hfarbe%)
       CASE 7    'Hilfsfunktion
         GOSUB Hilfe
       CASE ELSE
     END SELECT
WEND
CLOSE #DNr
COLOR 7, 0
CLS
PRINT "Programm beendet..."
END
```

```
'*********************************************************************
'* Fehlerbehandlung und Ereignisverfolgung                           *
'*********************************************************************

Fehlerbehandlung:
  SELECT CASE ERR
    CASE 6
       Fehler$="Überlauf!"
    CASE 14, 201
       Fehler$="unzureichender Speicherplatz!"
    CASE 24, 25, 68, 69
       Fehler$="Peripherie-, Schnittstellenprobleme!"
    CASE 27, 57
       Fehler$="Der Drucker ist nicht bereit!"
    CASE 51
       Fehler$="Interner Fehler!"
    CASE 70, 71, 72
       Fehler$="Probleme mit dem Laufwerk!"
    CASE 71
       Fehler$="Keine Diskette/Klappe offen!"
    CASE ELSE
       Fehler$="Fehler Nummer "+STR$(ERR)+" ist aufgetreten!"
  END SELECT
  CALL InfoZeile("Achtung: "+Fehler$, 25, vErr%, hErr%)
  CALL Warte
RESUME Start

Hilfe:
  CALL InfoZeile("Wählen Sie ein Thema, zu dem Sie Informationen"+_
                 " wünschen...", 25, hfarbe%, vfarbe%)
  CALL HinterGrund(vfarbe%, hfarbe%)
  Help$(1)="MENUE" : Help$(2)="DATEIWAHL"
  Help$(3)="DATENEINGABE" : Help$(4) = "AUSWERTUNG"
  Help$(5)="DRUCKER" : Help$(6)="GRAFIK"
  CALL Auswahl(Help$(), 6, hfarbe%, vfarbe%, X%)
  HilfeBegriff$=RTRIM$(Help$(X%))
  CALL Hintergrund(vfarbe%, hfarbe%)
  HNr = FREEFILE
  OPEN "HAUSHALT.HLP" FOR INPUT AS #HNr
  HAnz%=0
  WHILE NOT EOF(HNr)
    LINE INPUT #HNr, Zeile$
    IF INSTR(Zeile$, "["+HilfeBegriff$+"]") > 0 THEN
      WHILE Left$(Help$(HAnz%),1)<>"#"
        HAnz%=HAnz%+1
        LINE INPUT #HNr, Help$(HAnz%)
      WEND
    END IF
  WEND
  CALL Auswahl(Help$(), HAnz%-1, hfarbe%, vfarbe%, 0)
  CLOSE #HNr
  CALL Hintergrund(vfarbe%,hfarbe%)
RETURN
```

Listing 2.9.2: Das vollendete Hauptprogramm

Wir haben für unseren Index lediglich sechs Begriffe zu berücksichtigen, die nach der Sprungmarke *Hilfe* initialisiert werden. Mit der Routine *Auswahl* bringen wir diese Begriffe auf den Bildschirm. Sie sehen, wie vielseitig einsetzbar unsere allgemeine Auswahlroutine ist. Während der späteren Programmausführung stellt sich der Bildschirm wie in Bild 2.9.1 dar.

Bild 2.9.2: Der Hilfe-Index

Um den Hilfsbegriff zu ermitteln, spalten wir die Leerzeichen des gewählten Menüeintrages ab, die durch die Auswahlroutine angefügt wurden (*RTRIM$-Funktion*). Nachdem der Hilfsbegriff bekannt ist, können wir ihn in der sequentiellen Datei HAUSHALT.BAS suchen lassen. Beachten Sie, daß der Vergleich in unserer Routine nicht zwischen Klein- und Großschreibung unterscheidet. (Hilfsbegriffe des Index sind nach der Sprungmarke *Hilfe* und in der Hilfsdatei HAUSHALT.HLP in Großschrift definiert und werden auf Gleichheit überprüft.) Die Begriffe, die als Index definiert sind, müssen also unbedingt in Großbuchstaben definiert sein.

Über die Funktion *FREEFILE* ermitteln wir zunächst eine freie Dateinummer, die wir in unserer *OPEN*-Anweisung verwenden. Um die Datei zu lesen, ist unbedingt der *INPUT*-Modus zu wählen. Erinnern Sie sich daran, daß der *OUTPUT*-Modus Ihre Hilfsdatei zerstört und eine neue Eingabe der Datei erforderlich machen würde. Da unsere Datei recht klein ist und der komplette Lesevorgang nur wenig Zeit in Anspruch nimmt, werden wir die Daten bei jeder Hilfeanforderung erneut aus der Datei lesen. Der Lesevorgang selbst erfolgt, wie bereits im Informationsteil angeführt, mit Hilfe einer *WHILE-WEND*-Schleife unter Verwendung der *EOF*-Funktion.

Diese Schleife liest die Hilfsdatei so lange zeilenweise ein, bis der Hilfsbegriff gefunden wurde. Danach wird eine zweite *WHILE-WEND*-

Schleife aktiv, die das Datenfeld *Help$()* mit passenden Textzeilen zu diesem Hilfsbegriff aufbaut. Der Abbruch erfolgt in diesem Fall durch das Doppelkreuz in der Hilfsdatei. Auch wenn das Doppelkreuz erreicht wurde, wird der Rest der Datei bis zum Dateiende abgearbeitet. Dafür ist dann wieder die äußere *WHILE-WEND*-Schleife verantwortlich.

```
HAUSHALT Version 1.00 - Einnahmen-/Ausgabenüberschußrechnung

Dateiwahl-Hilfe:
Ist noch keine Arbeitsdatei angelegt, wird automatisch
die Datei HAUSHALT.DAT angelegt. Ansonsten können Sie
auch eine neue Datei erzeugen. Das Programm kann in
dieser Version maximal 10 Dateien verwalten.

Wählen Sie ein Thema, zu dem Sie Informationen wünschen...
```

Bild 2.9.3: Hilfstext zum Menüpunkt Dateiwahl

In den Bildern 2.9.2 und 2.9.3 sind spezielle Bildschirmausgaben zu sehen, die von unserer Hilfsfunktion ausgegeben werden. Auch hier verwenden wir zur Darstellung des Textes die Routine *Auswahl* – auch, wenn wir in diesem Fall keine Menüauswahl vornehmen. Der verschiebbare Balken kann dabei allerdings als Lesezeichen verwendet werden.

```
HAUSHALT Version 1.00 - Einnahmen-/Ausgabenüberschußrechnung

Grafik-Hilfe:
Die grafische Ausgabe besteht aus zwei Teilen. Zunächst werden
die monatlichen Salden in einem Liniendiagramm aufgetragen,
um die Gewinne und Verluste zwischen den Monaten eines Jahres
sehr schnell und anschaulich vergleichen zu können. In einem
Kreisdiagramm werden dann die Ausgabensumme und die Gewinnsumme
für den gesamten Buchungszeitraum dargestellt.

Wählen Sie ein Thema, zu dem Sie Informationen wünschen...
```

Bild 2.9.4: Hilfstext zum Menüpunkt Grafik

Da die Hilfsdatei jedesmal erneut geöffnet wird, müssen wir sie abschließend wieder schließen. Eine bereits geöffnete Datei darf nicht zweimal

geöffnet werden, da sonst der Laufzeitfehler 55 auftritt (»Datei bereits geöffnet«). Über *RETURN* kehren wir abschließend in die Hauptschleife des Hauptprogramms zurück. Mit diesen Anweisungen ist unser Programm fertiggestellt. In unserer nächsten Sitzung werden wir uns mit der Testphase und der Erstellung des Maschinenprogramms befassen.

Workshop

Checkliste

1. Erläutern Sie den sequentiellen Dateimodus!
2. Was sind die wesentlichen Unterschiede zwischen Direktzugriffsdateien und sequentiellen Dateien?
3. Was verstehen Sie unter Ereignisverfolgung?
4. Welche Möglichkeiten bietet der *SHELL*-Befehl von Power Basic?
5. Was bedeutet kontextsensitive Hilfe?
6. Wie können Sie vermeiden, daß Sie über das Ende einer sequentiellen Datei hinaus lesen und damit einen Laufzeitfehler erzeugen?

Ideen für eigene Übungen

- Erweitern Sie das Programm HAUSHALT so, daß Sie bereits vor der Ausführung des *SHELL*-Befehls die Möglichkeit haben, einen beliebigen Befehl für das Betriebssystem einzugeben. Verwenden Sie hierzu die in unserer Funktionsbibliothek erstellte Routine *Eingabe*!
- Optimieren Sie die geschachtelte *WHILE-WEND*-Konstruktion für die Hilfsfunktion soweit, daß der Lesevorgang jedesmal nach dem ersten Doppelkreuz beendet wird!
- Stellen Sie die Routine zur Ausgabe der Hilfsinformation so um, daß das Einlesen der gesamten Hilfsdatei nur einmal erfolgt. Verwenden Sie dazu ein Zeichenketten-Datenfeld, das beim Aufruf der Hilfsfunktion entsprechend der Datei durchsucht wird!
- Schreiben Sie ein kleines Programm und verwenden Sie die bereits vorhandene Hilfsdatei und die Anweisungen zur Hilfsfunktion aus dem Programm HAUSHALT. Lassen Sie mit Hilfe der Ereignisverfolgung über die Funktionstaste [F1] (Tastaturcode CHR$(59)) unseren Hilfe-Index ausgeben. Beenden Sie das Programm durch die Funktionstaste [F10] (Tastaturcode CHR$(68).

2.10 Zehnte Sitzung: Das Maschinenprogramm entsteht

Die Themen dieser Sitzung:

- Testphase
- Ausführbare Programmdateien
- Ausnutzung der Compiler-Optionen
- Ausnutzung der Linker-Optionen
- Programmweitergabe

Information

In dieser letzten Sitzung wollen wir abschließend aus unserem Quelltext, den wir in unseren Power-Basic-Editor eingegeben haben, ein eigenständig ablauffähiges Programm erzeugen. Das Programm soll aber nicht nur ohne das Power-Basic-Entwicklungssystem einsetzbar sein, sondern auch für einen entsprechenden Rechner in Geschwindigkeit und Programmgröße optimiert werden.

Die Testphase

Sie sollten, falls Sie Ihr Programm an Dritte weitergeben, zunächst eine Phase einlegen, in der Sie Ihr Programm ausgiebig testen. Auch wenn Ihr Programm auf den ersten Blick scheinbar korrekt arbeitet, kann es dennoch interne Fehler beinhalten. Die Testphase kann, je nach Programm, unterschiedlich gestaltet sein. Nehmen wir an, Sie haben ein Programm für mathematische Berechnungen programmiert. Kontrollieren Sie das Programm mit Beispielrechnungen, die Sie bereits per Hand berechnet haben, oder durch Rechnungen, deren Ergebnisse Sie genau kennen. Nehmen Sie unbedingt immer die Sonderfälle eines Problems, nicht nur Standardaufgaben. Lassen Sie Ihr Programm auch ruhig einmal von einer unbeteiligten Person ausführen. Sind im Programm spezielle Grenzen gesetzt, wie z.B. für eine maximale Datensatzanzahl, so überprüfen Sie, ob das Programm auch in diesem Grenzbereich korrekt arbeitet.

Natürlich kann auch die ausgiebigste Testphase nicht sämtliche Fehler ausräumen. Auch in kommerziellen Programmen treten hin und wieder Fehler auf.

Ausführbare Programmdateien

Das Betriebssystem kennt zwei Typen von ausführbaren Programmdateien. Da sind zunächst die COM-Dateien, die eine maximale Größe von 64 Kbyte im Speicher einnehmen dürfen, und zum anderen die EXE-Dateien, die den gesamten Arbeitsspeicher nutzen können. Power Basic erlaubt die Erstellung von EXE-Dateien aus Power-Basic-Quelltext, d.h., die Programme, die mit Power Basic erstellt werden, können den gesamten Arbeitsspeicher nutzen.

Die Arbeit, die das Programmentwicklungssystem leisten muß, ist hinter einer Oberfläche versteckt und läßt sich nur anhand der Menüeinträge erahnen. Ein Quelltext wird in mehreren Schritten übersetzt. Zunächst wandelt der Compiler das Programm in Maschinensprache um und fügt Sprungmarken für die Bibliotheksfunktionen ein. Der Linker löst im zweiten Schritt diese Marken durch die Verbindung mit der Laufzeitbibliothek auf. Sämtliche Arbeitsgänge verlaufen im Hintergrund. Über den Menüeintrag *Options* kann der Programmierer diese Arbeitsschritte weitreichend beeinflussen.

Compiler-Optionen

Über die Compiler-Optionen hat der Programmierer die Möglichkeit, den generierten Code zu beeinflussen. Werden die Optionen nicht korrekt gesetzt, kann es durchaus sein, daß das Programm auf Ihrem Rechner nicht mehr lauffähig ist.

Die Möglichkeit, die Einstellungen für den Compiler zu ändern, sollten Sie insbesondere nutzen, wenn Sie über einen Coprozessor oder einen 80286- oder höheren Hauptprozessor verfügen. Dann wird das erzeugte Maschinenprogramm kleiner in der Programmgröße und schneller in der Ausführungsgeschwindigkeit.

Zwei weitere wichtige Optionen wollen wir an dieser Stelle nennen. Arbeiten Sie mit Datenfeldern, so können Sie die Standarddimensionierung mit der Option *Array auto-dim On* auf zehn Feldelemente festlegen. Diese Möglichkeit haben wir auch im Programm HAUSHALT genutzt.

Bedenken Sie, wenn Datenfelder mehr als zehn Elemente enthalten sollen, ist unbedingt eine *DIM*-Anweisung erforderlich.

Wenn Sie die Option *Control-Break testing On* einsetzen, können Sie Ihr Programm bei sämtlichen Programmein- und -ausgaben durch die Tastenkombination [Ctrl] + [Break] bzw. [Strg] + [Untbr] abbrechen und zur Betriebssystemumgebung zurückgelangen. Ohne diese Option ist ein vorzeitiger Programmabbruch nicht möglich.

Linker-Optionen

Mit Hilfe der Linker-Optionen können Sie die Programmgröße bei der Code-Erzeugung reduzieren. Wie bereits erwähnt, beinhaltet Power Basic eine sogenannte Laufzeitbibliothek, die in jedes Maschinenprogramm mit eingebunden wird. Hierin sind die Standardanweisungen wie z.B. *LINE*, *LPRINT* usw. in Maschinensprache enthalten. Nun ist es allerdings nicht notwendig, die Grafikroutinen in eine reine textorientierte Anwendung, die Druckerausgabe in einem rein bildschirmorientierten Programm oder aber Routinen für die serielle Schnittstelle in ein Programm einzubauen, das diese Schnittstelle gar nicht nutzt. Der Programmierer hat nun die Möglichkeit, die Optionen so zu setzen, daß nicht benötigte Teile der Laufzeitbibliothek in das Programm nicht mit eingebunden werden.

Programmweitergabe

Bei jeder Software sind spezielle rechtliche Bestimmungen zu berücksichtigen, die an dieser Stelle kurz behandelt werden sollen. Nachdem Sie Power Basic rechtmäßig erworben, und sich registrieren ließen, können Sie Programme mit Power Basic entwickeln. Beachten Sie, daß eine Weitergabe dieses Entwicklungssystemes nicht zulässig ist. Power Basic erlaubt Ihnen neben der Erstellung von Quelltexten die Erzeugung ausführbarer Programmdateien. Handelt es sich bei den Quelltexten um Ihr eigenes geistiges Eigentum, können Sie entscheiden, ob Sie den Quelltext oder nur das ausführbare Programm weitergeben.

In der Regel werden Sie den Quelltext behalten und nur das Maschinenprogramm vertreiben. Auf diesem Wege können Sie verhindern, daß andere Programmierer Ihren Quelltext ungewollt verändern oder auch weiterentwickeln.

Praxis

Wir wollen nun unser Programm HAUSHALT.BAS in das ausführbare Maschinenprogramm übersetzen. Dabei gehen wir davon aus, daß unser Rechner entweder über einen 80286-, 80386- oder 80486-Hauptprozessor verfügt und mit einer VGA-Grafikkarte ausgestattet ist.

Der Compiler

Wir befinden uns immer noch in der integrierten Entwicklungsumgebung von Power Basic. Um den Editor zu verlassen und wieder in die Menüleiste zu gelangen, betätigen wir die Funktionstaste [F10]. Anschließend öffnen wir über die Menüfolge *Options* → *Compiler* das Auswahlfenster für die Compiler-Optionen (Bild 2.10.1).

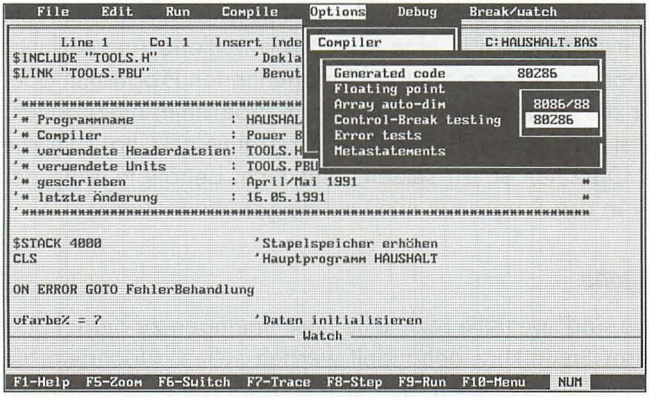

Bild 2.10.1: Compiler-Optionen

An dieser Stelle wählen wir *Generated Code* [Return] und anschließend aus einem weiteren Fenster den *80286*-Eintrag aus. Alle übrigen Optionen sind in der Regel standardmäßig gesetzt und brauchen nicht geändert zu werden. Wenn wir davon ausgehen, daß unser Programm fehlerfrei läuft, können wir zur Reduzierung der erzeugten Code-Größe die Fehlertests (Error test) ausschalten. Ansonsten generiert der Compiler zusätzlichen Programmcode für spezielle Prüfroutinen.

Der Linker

Wie wir bereits erwähnt haben, ist unser Programm bei den Grafikausgaben speziell an den VGA-Modus angepaßt. Da wir daher keine Unterstützung anderer Grafikkarten in unserem Programm benötigen,

werden wir nun die Linker-Optionen entsprechend umstellen. Dazu rufen wir über die Menüfolge *Options* → *Linker* das Fenster mit den Linker-Optionen auf. Wählen wir nun den Eintrag *Video cards*, können wir je nach Bedarf Routinen für bestimmte Grafikkarten aktiv oder inaktiv schalten. In unserem Fall setzen wir die Karten *CGA*, *EGA* und *Herc* auf *Off* (Bild 2.10.2).

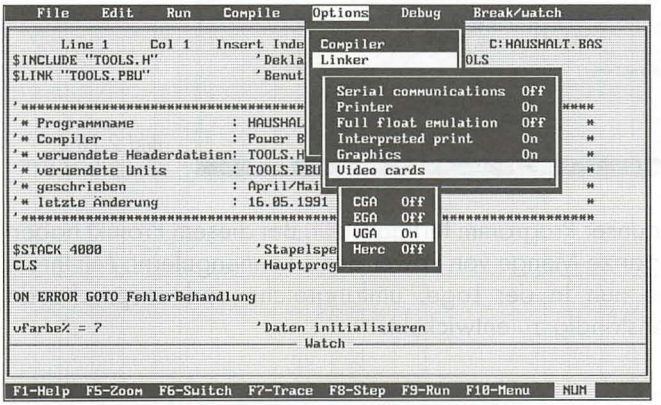

Bild 2.10.2: Linker-Optionen

Im Anschluß daran sollte auch der Eintrag *Serial communications* auf *Off* gestellt werden, da in unserem Programm HAUSHALT.EXE keine Anweisungen für die serielle Schnittstelle benötigt werden.

Das Maschinenprogramm entsteht

Nachdem sowohl die Optionen des Compilers als auch des Linkers eingestellt sind, können wir endlich das endgültige Maschinenprogramm erzeugen lassen. Dazu müssen wir zunächst kontrollieren, ob das Maschinenprogramm nicht im Speicher, sondern als EXE-Datei erstellt wird (Bild 2.10.3). Über *Compile* → *Destination* → *EXE file* können wir, falls nötig, diese Umstellung vornehmen. Anschließend starten wir den Compiler mit [Alt] + [F9]. In einem Bildschirmfenster wird der aktuelle Arbeitsstand und die fehlerfreie Übersetzung nach der Fertigstellung angezeigt. Danach können Sie Power Basic mit *File* → *Quit* oder der Tastenkombination [Alt] + [X] verlassen. Achten Sie darauf, daß Sie den letzten Arbeitsstand der Datei HAUSHALT.BAS spätestens bei der Sicherheitsabfrage von Power Basic speichern.

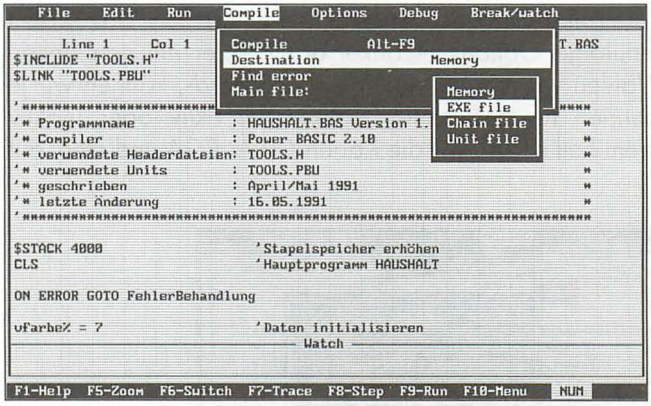

Bild 2.10.3: Compile to EXE

Damit haben wir unser Programm fertiggestellt. Bei diesem Projekt haben Sie bereits eine ganze Menge von Power Basic kennengelernt. Mit Ihrem Wissen sind Sie nun in der Lage, umfangreiche und ansprechende Programme eigenständig zu entwickeln.

Workshop

Checkliste

1. Wozu dient die Testphase bei der Programmentwicklung?
2. Welche Möglichkeiten bieten sich über die Compiler-Optionen?
3. Welche Möglichkeiten ergeben sich durch die Linker-Optionen?
4. Was sind ausführbare Programmdateien?
5. Erläutern Sie die Vorgehensweise zur Erstellung eines Maschinenprogramms!

Ideen für eigene Übungen

- Erstellen Sie ein Programm, das einen kurzen Text auf dem Bildschirm ausgibt. Das Programm soll durch Betätigen einer beliebigen Taste beendet werden. Übersetzen Sie anschließend dieses Programm in eine ausführbare Programmdatei. Stellen Sie die Compiler- und Linker-Optionen so ein, daß die Programmgröße minimal wird, das Programm aber dennoch auf Ihrem Rechner lauffähig bleibt! Sie können hierzu mehrere EXE-Dateien erzeugen und die Programmgröße über das Betriebssystem-Kommando *DIR* direkt vergleichen.

Kapitel 3: Know-how

In unserem dritten Kapitel wollen wir uns mit einigen Besonderheiten von Power Basic beschäftigen. Neben den Fehlermeldungen werden wir uns mit speziellen Programmierthemen, dem Kommandozeilen-Compiler und nützlicher Zusatzsoftware befassen.

3.1 Fehlerbeseitigung

Wie wir in Kapitel 2.9 bereits gelernt haben, erkennt Power Basic selbständig eine Vielzahl von Fehlern. Unterschieden wird zwischen den Laufzeitfehlern und den Syntaxfehlern. Werden die Syntaxfehler vom Compiler direkt beim Übersetzungsvorgang erkannt, sind die Laufzeitfehler nur schwer zu beseitigen. Zunächst wollen wir aber bei den Compiler- bzw. Syntaxfehlern bleiben. Die nachfolgende Liste gibt zusammengefaßt die wichtigsten Fehlermeldungen des Compilers wieder.

401–405: Ausdruck oder Befehlsfolge zu komplex

Zerlegen Sie den Ausdruck oder die Befehlsfolge in mehrere einfache Teile, und reduzieren Sie die Verschachtelungen.

406–410: Unzureichender Variablen- oder Programmspeicher

Arbeiten Sie mit Units/Include-Dateien, zerlegen Sie das Programm in mehrere kleine Teile. Benutzen Sie unter Umständen den Kommandozeilen-Compiler und verwenden Sie weniger Variablen, Konstanten usw.

411–421: einfache Syntaxfehler

Wahrscheinlich haben Sie einen Schreibfehler gemacht; der Compiler kann die Quelltextzeile nicht korrekt auswerten.

422–433: falscher Datentyp

Sie haben wahrscheinlich ein unter Power Basic reserviertes Wort als Variablenname verwendet bzw. in einer Anweisung einen falschen Datentyp für eine Variable verwendet.

434–459: fehlerhafte Blockstruktur bzw. benutzerdefinierte Anweisung

Sie haben nicht die vollständige Syntax eines Blockbefehls, einer Funktion oder eines Unterprogramms verwendet! Erinnern Sie sich, daß z.B. zu jedem *SUB* ein *END-SUB* und zu jedem *SELECT-CASE* ein *END-SELECT* gehört!

460–470: Sprachelement nicht definiert bzw. doppelt vorhanden!

Ändern Sie doppelte Namen oder Sprungmarken ab und kontrollieren Sie, ob diese bereits im Programm eingeführt sind.

471–492: falscher Befehl

Sie haben einen Befehl verwendet, der an der bestimmten Stelle im Programm nicht zulässig ist! Eventuell haben Sie unzulässige Zeichen in Variablen- oder Unterprogrammnamen verwendet oder ein Datenfeld zu groß dimensioniert.

493–497: Dateifehler

Wahrscheinlich haben Sie bei einer *$INCLUDE*- bzw. *$INLINE*- oder *$LINK*-Anweisung einen falschen Dateinamen angegeben.

498–516: Dateimodulfehler

Eine Anweisung in einer Unit oder einem Assemblermodul ist nicht korrekt bzw. in Verbindung mit dem aktuellen Hauptprogramm nicht zulässig.

601–606: Interner Fehler

Fehler im Laufzeitsystem von Power Basic! Sie haben keine Möglichkeit, diese Fehler zu beseitigen!

Entdeckt der Compiler einen dieser Fehler, wird der Übersetzungsvorgang abgebrochen und der Cursor wird im Editor an die Fehlerstelle positioniert. So haben Sie direkt die Möglichkeit, unmittelbar eine Korrektur vorzunehmen. Eine Vielzahl der hier genannten Fehler ergibt sich durch Tippfehler. Korrekturen sind daher normalerweise sehr schnell durchführbar.

Auf eine erneute Auflistung der Laufzeitfehlermeldungen wollen wir an dieser Stelle verzichten. Nähere Informationen hierzu können Sie Kapitel 2.9 entnehmen. Das Problem bei den Laufzeitfehlern ist, daß diese erst nach der Programmerstellung auftreten und daher nur schwer auffindbar sind. In unserem Programm HAUSHALT fangen wir eventuell auftretende Fehler ab und geben eine aussagekräftige Fehlermeldung aus. In komplexeren Programmen müßte diese Fehlerliste natürlich erweitert werden. Verzichten wir auf eine Fehlerbehandlungsroutine, wird das Programm beim Auftreten eines Laufzeitfehlers abgebrochen. Innerhalb der Oberfläche wird der Textcursor wieder an die Fehlerstelle positioniert.

Der Programmzähler

Da Sie Ihr Programm später im Normalfall in einer eigenständig ausführbaren Programmfassung nutzen werden, haben Sie eine zusätzliche Möglichkeit, um Fehler zu lokalisieren. Nachdem in einem Maschinenprogramm ohne Fehlerbehandlungsroutine ein Laufzeitfehler auftritt, wird das Programm beendet und eine Fehlermeldung auf der Betriebssystemebene in der nachfolgenden Form ausgegeben:

```
Error 75 at pgm-ctr: 1334
```

Diese Fehlermeldung ermöglicht es, die Stelle im Quelltext zu ermitteln, die den Fehler verursacht hat. Dazu starten Sie die Power-Basic-Entwicklungsumgebung und laden den Quelltext zu Ihrem Maschinenprogramm. Wählen Sie anschließend die Menüfolge *Compile* → *Destination* →

EXE-File, damit der Compiler die Beziehung zwischen dem Maschinenprogramm und dem Quelltext herstellen kann. Danach können Sie mit *Compile → Find Error* ein Eingabefenster öffnen, in das Sie die Fehlernummer eingeben können (Bild 3.1.1). Wenn Sie diese Eingabe mit [Return] quittieren, wird der Textcursor genau in die Quelltextzeile positioniert, in der der Fehler im Maschinenprogramm aufgetreten ist.

```
File   Edit   Run   Compile   Options   Debug   Break/watch
    Line 1     Col 1      Compile          Alt-F9                  AS
'*****************************  Destination                EXE file  *
'* UNIT: TOOLS.BAS             Find error                           *
'* allgemein einsetzbare        Error Address                       *
'*****************************   1334                               *

SUB Ausuahl (Text$(), Anzahl%, vfarbe%, hfarbe%, Wahl%) PUBLIC
  COLOR vfarbe%, hfarbe%          'Fensteraufbau
  Breite% = 18
  FOR x% = 1 TO Anzahl%           'max. Stringlänge
    IF LEN(Text$(x%)) > Breite% THEN Breite% = LEN(Text$(x%))
  NEXT x%
  FOR y% = 1 TO Anzahl%           'einheitl. Stringlänge
    Text$(y%) = SollLen$(Text$(y%), Breite%)
  NEXT y%
  Breite% = Breite% + 2           'Fenster
  CALL Fenster(5, 5, Breite%, Anzahl%, vfarbe%, hfarbe%)
  FOR x% = 1 TO Anzahl%
    IF x% = 1 THEN
                          Watch
Alt: F1-Last help  F3-Pick  F5-User screen  F6-Swap  F9-Compile  X- NUM
```

Bild 3.1.1: Die Fehlerlokalisierung

Der Quelltext-Debugger

Um weitere Fehler zu entdecken, die weder zu einem Compiler- noch zu einem Laufzeitfehler führen, stellt Ihnen Power Basic einen Quelltext-Debugger zur Verfügung. Stellen Sie sich vor, Sie haben nach Ihrem Ermessen ein korrektes Programm erstellt, das Datensätze nach bestimmten Kriterien durchsucht. Die Ergebnisse sind aber seltsamerweise nicht korrekt, da nur ein kleiner Teil der gesuchten Datensätze wirklich gefunden wird. Aber wo und wie den Fehler finden? An einem kleinen Beispiel wollen wir die Möglichkeiten des Debuggers vorstellen.

Beispielprogramm Debugger:

```
FOR i%=1 TO 100
  z%=z%+1
  y%=y%+2
  x%=y%+z%
NEXT i%
```

Mit dem Debugger haben Sie die Möglichkeit, Variablenwerte zu verfolgen und zu verändern oder spezielle Haltepunkte in Ihr Programm einzubauen. Das Programm können Sie blockweise oder auch im Einzelschrittmodus ausführen lassen. Was das bedeutet, wollen wir an unserem

obigen Beispiel durchsprechen. Zunächst müssen Sie das Programm in den Editor der Benutzeroberfläche eingeben. Anschließend können wir es ausführen. Da das Programm keine Ausgaben auf dem Bildschirm macht, ist eine Verfolgung der Werte zunächst nicht möglich.

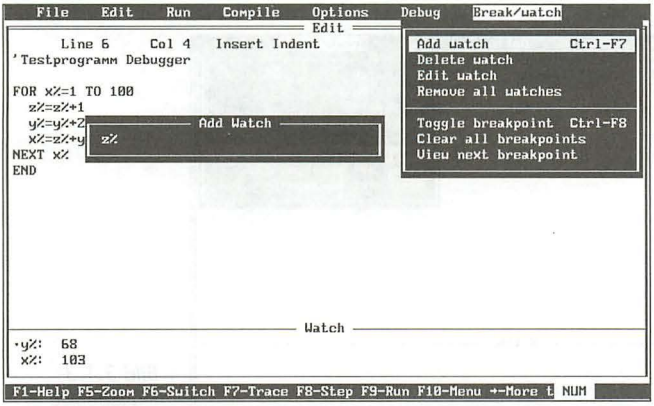

Bild 3.1.2: Variablen verfolgen

Um das zu ändern, lassen wir uns innerhalb der Oberfläche die Variablen *x%*, *y%* und *z%* anzeigen. Um eine Variable im sogenannten Watch-Fenster, das im unteren Bildschirmbereich angesiedelt ist, anzeigen zu lassen, wechseln wir mit [F10] zunächst vom Editor in die Menüauswahl. Mit *Break/watch* → *Add watch* oder dem Tastenbefehl [Strg] + [F7] können wir nun ein Fenster öffnen, in das ein beliebiger Variablenname eingebbar ist. Das ist in unserem Falle zunächst *x%*. Quittieren wir diese Eingabe mit [Return], erscheint im Watch-Fenster die Variable *x%* mit dem eventuell bereits zugewiesenen Variablenwert (Bild 3.1.2).

Sobald wir *x%* als zu verfolgende Variable definiert haben, machen wir dasselbe mit den Variablen *y%* und *z%*. Führen wir nun das Programm aus, haben wir zwar den Endwert der Variablen in unserem Variablenfenster stehen, aber von Zwischenwerten ist keine Spur. Immerhin werden die Werte in unserem Programm innerhalb einer *FOR-NEXT*-Schleife genau hundertmal geändert. Aber auch hierfür bietet der Debugger entsprechende Funktionen an.

Um Zwischenwerte, z.B. in einer Programmschleife oder an einer beliebigen Programmstelle, anzeigen zu lassen, haben wir die Möglichkeit, Breakpoints bzw. sogenannte Haltepunkte zu setzen. Die Programmausführung wird später genau an dieser Stelle angehalten. Ein einmal

gestopptes Programm braucht nicht erneut gestartet zu werden, es kann genau an dieser Haltestelle wieder fortgesetzt werden. Innerhalb eines Programms können beliebig viele Haltepunkte gesetzt werden. In unserem Beispiel brauchen wir lediglich einen Haltepunkt innerhalb der Schleifenkonstruktion.

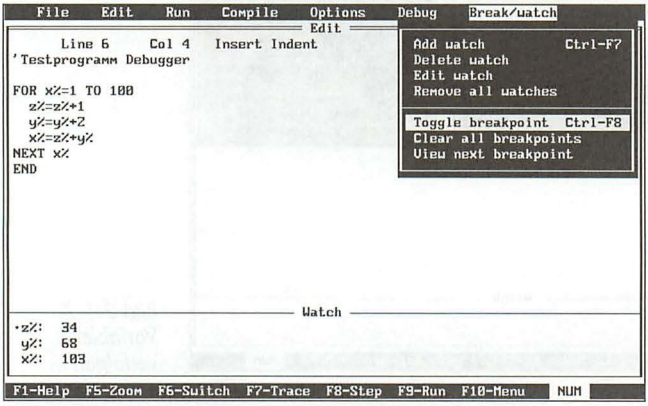

Bild 3.1.3: Haltepunkte setzen

Um einen Haltepunkt zu definieren, positionieren wir zunächst den Textcursor im Editor genau auf die Zeile, in der der Haltepunkt benötigt wird. Anschließend setzen wir den Breakpoint mit [F10] *Break/watch* → *Toggle Breakpoint* bzw. der Tastenkombination [Strg] + [F8] (Bild 3.1.3).

Starten wir nun unser Programm erneut, wird genau beim Haltepunkt die Programmausführung unterbrochen. Außerdem sind die Zwischenwerte der Variablen im *Watch*-Fenster zu erkennen. Mit [F9] können Sie nun das Programm bis zum nächsten Haltepunkt ausführen lassen. Jetzt können Sie jeden Zwischenwert der Variablen *x%*, *y%* und *z%* verfolgen.

An manchen Programmstellen kann es zudem erforderlich sein, ein Programm nicht bis zu einem bestimmten Punkt ablaufen zu lassen, sondern schrittweise auszuführen. Wird z.B. eine Zeichenkette in einem mehrzeiligen Block aufgebaut, ist es nicht notwendig, in jeder Zeile einen Haltepunkt einzufügen, einer zu Beginn des Blocks genügt. Anschließend setzen Sie das Programm mit der Funktionstaste [F7] schrittweise fort.

Ergeben sich für eine Variable in einem solchen Fall falsche Zwischenwerte, wäre es zum Teil sinnvoll, den Rest der Anweisungen auf Korrektheit zu überprüfen. Dazu müßte natürlich zunächst der falsche Zwischenwert korrigiert werden. Auch hierfür stellt Power Basic eine komfortable Mög-

lichkeit zur Verfügung. Über [F10] *Debug → Evaluate* bzw. mit der Funktionstaste [F4] können Sie Variablenwerte während der Programmausführung ändern. Damit können Sie die Auswirkungen unterschiedlicher Werte für Variablen in einem Programm testen, ohne den Quelltext bereits verändern zu müssen (Bild 3.1.4 und 3.1.5).

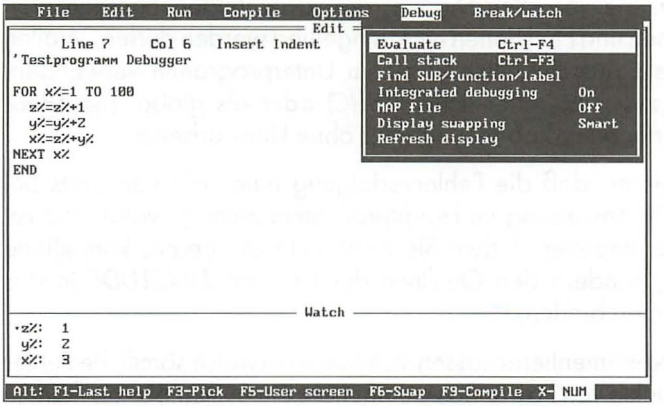

Bild 3.1.4: Variablenmanipulation

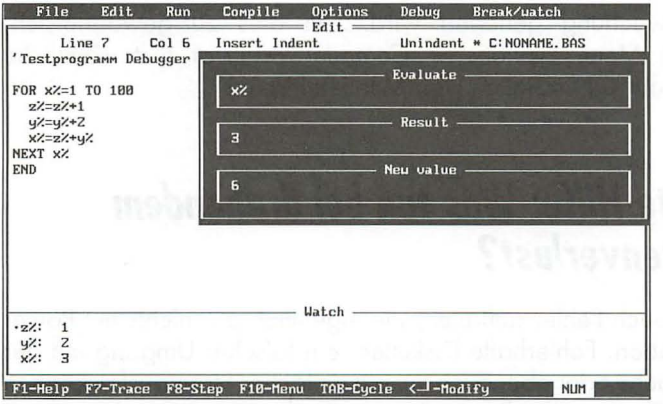

Bild 3.1.5: Variablenwerte ändern

Nun haben Sie eine Vielzahl von Möglichkeiten kennengelernt, um Fehler, die in der Programmierung mit Power Basic auftreten können, zu beseitigen. Einen Großteil der Fehlersuche erledigt Power Basic für Sie, für nicht erkennbare Fehler stellt die Entwicklungsumgebung hervorragende Werkzeuge bereit.

Bekannte Programm-Mängel und deren Umgehung

Da es derzeit nach menschlichem Ermessen nicht möglich ist, Software-Programme fehlerfrei zu erstellen, sind sicherlich auch in Power Basic Fehler vorhanden. Im großen und ganzen erweist sich der Compiler allerdings, im Gegensatz zu seinem Vorgänger Turbo Basic, als sehr stabil. Beachten Sie vor allen Dingen, daß Datenfelder innerhalb von Unterprogrammen und Funktionen nicht eingeführt werden dürfen. Wollen Sie ein Datenfeld ausschließlich in einem Unterprogramm verwenden, müssen Sie es zuvor als öffentlich (*PUBLIC*) oder als global (*SHARED*) deklarieren. Je nachdem, ob Sie mit oder ohne Units arbeiten.

Beachten Sie ferner, daß die Fehlerverfolgung innerhalb von Units bei einer *ON-ERROR*-Anweisung im Hauptprogramm nicht gewährleistet ist. Sie können dies umgehen, indem Sie nicht mehr die bereits kompilierte Unit mit *$LINK*, sondern den Quelltext der Unit mit *$INCLUDE* in das Hauptprogramm einbinden.

Mit ein wenig Experimentieren lassen sich normalerweise sämtliche Fehler und Schwachpunkte von Power Basic umgehen. Ein Systemabsturz kommt in der Regel während der Programmierung nicht vor, wenn Code für die Programmüberwachung generiert wird. Um dies zu gewährleisten, müssen Sie im Menü *Options* → *Compiler* → *Error tests* sämtliche Menü-Einträge auf *ON* schalten (vgl. Menü-Referenz).

3.2 Erste Hilfe: Was tun bei drohendem Datenverlust?

Leider können auch Fehler auftreten, die eigentlich gar nichts mit Power Basic zu tun haben. Fehlerhafte Disketten, ein falscher Umgang mit der Benutzeroberfläche oder aber auch nur ein einfacher Stromausfall können ungewollt zum Datenverlust führen. Wer will schon gerne ein Programm zweimal eintippen? Die sicherste Möglichkeit, die Gefahren eines Datenverlustes so gering wie möglich zu halten, ist die regelmäßige Datensicherung. Dabei sind insbesondere die Quelldateien wichtig, aus denen Sie sich später ja wieder ein Maschinenprogramm erzeugen lassen können. Gewöhnen Sie sich an, nicht nur eine Kopie Ihres Programms auf der Festplatte zu erzeugen, sondern legen Sie die Quelltexte auch auf Diskette ab. Bedenken Sie, nicht jede Festplatte lebt ewig. Und ist eine

Festplatte erst einmal defekt, so kann Ihnen keiner mehr garantieren, daß Sie Ihre Daten jemals wiederverwenden können.

Sollte Ihnen einmal eine Arbeitsdatei zerstört worden sein, so können Sie versuchen, die entsprechende BAK-Datei in den Editor zu laden, die standardmäßig als Sicherungsdatei gespeichert wird. Hat Ihre Datei den Namen TEST.BAS, so trägt die Sicherungsdatei den Namen TEST.BAK. Ist dies nicht möglich, können Sie auf Ihre Diskettenkopie zurückgreifen.

3.3 Kniffe, Tips und andere Spezialitäten

An dieser Stelle wollen wir einige Tips und Tricks zu Power Basic vorstellen. Sie werden sehen, auch für die professionelle Programmierung ist Power Basic geeignet.

Programmierung für Profis

Nun wollen wir uns einigen speziellen Programmierthemen widmen, die die Leistungsfähigkeit von Power Basic unter Beweis stellen.

DOS-Piping und SHELL

Über den *SHELL*-Befehl können beliebige DOS-Befehle oder Programme, aus einem Power-Basic-Programm heraus, ausgeführt werden. Programme, die aufgerufen werden, zerstören durch Bildschirmausgaben zum Teil die eigene Programmoberfläche. Um dies zu verhindern, verwenden wir das sogenannte DOS-Piping, die Dateiumleitung des Betriebssystems. Wollen wir Ausgaben unterdrücken, stellt DOS die Einheit *NUL* bereit. Um z.B. mit *SHELL* ein Inhaltsverzeichnis auszugeben, das nicht auf dem Bildschirm erscheint, verwenden wir den nachfolgenden Befehl.

```
SHELL "DIR > NUL"
```

Ebenso könnten wir das Inhaltsverzeichnis in eine Datei speichern, z.B. mit dem Namen TEST.BAS.

```
SHELL "DIR > TEST.BAS"
```

Diese Datei wird im ASCII-Format abgelegt und könnte von Power Basic im sequentiellen Dateimodus direkt bearbeitet werden. Dieses Prinzip der Dateiumleitung ist mit einer Vielzahl von kommandozeilenorientierten Programmen möglich (z.B. Dienstprogramme des Betriebssystems).

Interrupt-Programmierung

Über die Interrupt-Programmierung können sämtliche DOS- und BIOS-Funktionen in einem Power-Basic-Programm verwendet werden. Aber auch Funktionen aus Treibern (z.B. Maus-Treiber, EMS-Treiber) können über diese Systemaufrufe angesprochen werden. Der Interrupt-Aufruf erfolgt nach einem einheitlichen Schema.

Schema für Interrupt-Aufrufe:

1. Falls erforderlich, Eingaberegister mit REG-Befehl setzen.
2. Interrupt mit CALL INTERRUPT aufrufen.
3. Eventuell Rückgabewerte aus den Registern mit der REG-Funktion lesen.

Wir wollen an dieser Stelle ein Unterprogramm vorstellen, das nach diesem Schema programmiert ist und auf den Maus-Treiber zugreift. Auch hierfür verwenden wir das oben vorgestellte Schema für Interrupt-Aufrufe. Wir haben diese spezielle Funktion gewählt, da sowohl Eingabe- als auch Rückgabewerte behandelt werden müssen. Außerdem sind die Werte direkt auswertbar und müssen nicht genauer entschlüsselt werden.

```
SUB MausTest (MausStatus%, AnzahlTasten%)
  REG 1, &H0  'Eingaberegister
  CALL INTERRUPT &H33  'MAUS-Interrupt
  MausStatus%=REG(1)
  AnzahlTasten%=REG(2)
END SUB
```

Tips und Tricks zu Power Basic

An dieser Stelle wollen wir Ihnen einige Tips geben, die Ihnen ein noch effektiveres Arbeiten mit Power Basic ermöglichen.

Drucken von Programmen

Power Basic beinhaltet leider keinen Menüeintrag, der das Ausdrucken eines Programms oder eines Programmteils erlaubt. Innerhalb des Editors können Sie über den Tastenbefehl [Strg] + [K] [P] einen markierten Block zum Drucker senden. Ist kein Block markiert, wird der gesamte Quelltext zum Drucker gesendet. Aber auch die Menüs erlauben das Ausdrucken einer Quelldatei. Wählen Sie *File → Write To* und geben Sie als Dateinamen PRN ein. In diesem Fall wird der gesamte im Editor befindliche Quelltext auf dem Drucker ausgegeben. PRN ist für MS-DOS der

Gerätename für den Drucker und als Dateiname daher unzulässig. Ebenso lassen sich auch einzelne Teile eines Programms ausdrucken. Markieren Sie den Blockanfang mit [Strg] + [K] [B] und das Blockende mit [Strg] + [K] [K]. Anschließend wählen Sie [Strg] + [K] [W], um einen Block in eine Datei zu schreiben. Hier geben Sie wieder PRN als Gerätenamen an.

Übernahme von GW-Basic-Programmen

Power Basic bietet Ihnen die Möglichkeit, Quelltexte, die ursprünglich für den GW-Basic- bzw. BasicA-Interpreter entwickelt worden sind, zu übernehmen und in Maschinensprache zu übersetzen. Dabei ist allerdings zu berücksichtigen, daß der Interpreter die Programme standardmäßig im sogenannten Token-Format, einem verschlüsselten Dateiformat, speichert. Diese Programme können erst von Power Basic benutzt werden, wenn sie vom Interpreter mit dem nachfolgenden Befehl gespeichert wurden.

Allgemein:

```
SAVE Dateiname ,A
```

Beispiel:

```
SAVE "TEST.BAS",A
```

Die Option A sorgt in unserem Beispiel dafür, daß die Datei TEST.BAS im ASCII-Format abgespeichert wird. Im Anschluß daran können Sie die Datei in den Editor von Power Basic laden und notwendige Änderungen durchführen.

Schreiben eines Startprogramms

Wenn Sie ein ausführbares Maschinenprogramm erstellen, wird, je nach Linker-Optionen, ein mehr oder weniger großer Teil der Laufzeitbibliothek mit eingebunden. Haben Sie mehrere Dateien auf einer Diskette oder der Festplatte, liegen diese Routinen mehrfach vor. Um wertvollen Speicherplatz auf Ihrem Festspeicher zu sparen, bietet es sich an, ein eigenes Laufzeitmodul zu verwenden, das nur einmalig vorhanden sein muß. Power Basic bietet hierzu eine besonders einfache Lösung an.

Zunächst erstellen Sie sich ein Hauptprogramm, das im Prinzip nichts weiter macht, als die Programmausführung an ein Chain-Modul zu übergeben. Mit einer kleinen Fehlerbehandlung und kurzem Hilfstext könnte ein solches Programm wie in Listing 3.3.1 aussehen.

```
ON ERROR GOTO Fehler
IF COMMAND$="" THEN
   PRINT "PBRUN - Power Basic 2.10 - Laufzeitmodul"
   PRINT "Programmstart: PBRUN Name"
 ELSE
   CHAIN COMMAND$+".PBC"
END IF
END

Fehler:
  PRINT "Datei kann nicht ausgeführt werden!"
RESUME NEXT
```

Listing 3.3.1: Das Laufzeitmodul PBRUN.BAS

Übersetzen Sie dieses Programm anschließend in eine ausführbare Programmdatei und beachten Sie, daß sämtliche Laufzeitroutinen eingebunden werden (alle Linker-Optionen auf *ON*). Damit haben Sie im Prinzip nichts anderes als eine »ausführbare« Laufzeitbibliothek erstellt. Mit *CHAIN* wird die Kontrolle des Programms an ein *CHAIN*-Modul übergeben, das beim Programmstart auf der Kommandozeilenebene mit angegeben wurde. Der *CHAIN*-Befehl sorgt dafür, daß die Laufzeitbibliothek im Speicher verbleibt und ein Overlay-Modul nachgeladen wird. Das Modul selbst kann nun auf die Laufzeitbibliothek verzichten. Wir übersetzen unsere Programme nun nur noch in *CHAIN*-Module (PBC-Dateien). Um diese Dateien später ausführen zu können, verwenden wir folgenden Aufruf:

```
PBRUN Programmname
```

Da unser Programm PBRUN.BAS ein Suffix automatisch anhängt, brauchen wir keines beim Dateinamen angeben. Nehmen wir an, unser Programm soll lediglich den Text »Hallo Welt!« auf dem Bildschirm ausgeben. Dazu benötigen wir nur eine Quelltextzeile.

```
PRINT "Hallo Welt!"
```

Diese übersetzen wir in ein *CHAIN*-Modul und starten es anschließend mit dem nachfolgenden Befehl.

```
PBRUN WELT
```

Das *CHAIN*-Modul belegt nun weniger als 400 Byte, da keine Laufzeitbibliothek mehr eingebunden werden muß. Wenn Sie so bei all Ihren Programmen vorgehen, werden Sie auf Dauer sehr viel Speicherplatz auf Ihren Disketten oder Ihrer Festplatte sparen.

Programmierung einer Fensterverwaltung

Die meisten Programme, die heute auf dem Markt erscheinen, verfügen über eine komfortable Benutzeroberfläche. Auch in unserem kleinen Programm HAUSHALT haben wir dies auf recht einfache Weise ermöglicht. Wie Ihnen aber vielleicht aufgefallen ist, überschreiben wir den Bildschirm jedesmal neu mit einem Hintergrund. Dadurch umgehen wir es, einen Fensteruntergrund in den Speicher zu sichern und nach dem Schließen des Fensters wieder zurückzuholen. Durch spezielle Power-Basic-Befehle können Routinen, die den Bildschirm sichern bzw. zurückholen, sehr schnell realisiert werden.

```
SUB Bildschirm(Karte$, Mode%)
  STATIC BildschirmInhalt$
  IF Karte$="HGC" THEN 'Hercules-Grafik
      DEF SEG=&HB000
    ELSE                 'sonstige Farbgrafik
      DEF SEG=&HB800
  END IF
  IF Mode%=1 THEN        'sichern
      BildschirmInhalt$=PEEK$(0,4000)
    ELSE                 'zurückschreiben
      POKE$ 0,BildschirmInhalt$
  END IF
  DEF SEG
END SUB
```

Listing 3.3.2: Die Routine Bildschirm

Das hier angeführte Unterprogramm sichert höchstens einen Bildschirm. Ist *Mode%=1*, wird gesichert, ansonsten wird der Bildschirminhalt restauriert. Wenn Sie mehrere Bildschirme sichern wollen, könnten Sie die einzelne Variable *BildschirmInhalt$* durch ein Stringarray ersetzen. In diesem Fall müßten Sie eine Fensternummer in der Parameterliste übergeben. Beachten Sie, daß die Zeichenkettenvariable *BildschirmInhalt$* eine statische Variable ist und daher der Wert bis zum nächsten Unterprogrammaufruf erhalten bleibt.

```
'Demo des Unterprogramms Bildschirm

CLS                        'Bildschirm löschen
COLOR 7,1                  'blau/weiß
FOR x%=1 TO 25             'Bildschirm aufbauen
  PRINT STRING$(80, "#");
NEXT x%
CALL Bildschirm("",1)      'sichern
COLOR 7,0                  'schwarz/weiß
CLS                        'Bildschirm löschen
```

```
PRINT "bitte Taste drücken..."
WHILE INKEY$="":WEND      'auf Tastendruck warten
CALL Bildschirm("",0)     'Bildschirm restaurieren
```

Listing 3.3.3: Bildschirm-Testprogramm

Mit dem Beispielprogramm aus Listing 3.3.3, welches hier für eine Farbgrafikkarte und einen Farbmonitor formuliert ist, können Sie die Routine zum Sichern und Zurückholen von Bildschirminhalten testen. Mit dieser Grundroutine und den Grundroutinen zur Fensterverwaltung aus dem Programm HAUSHALT können Sie sich nun eine umfangreiche Fensterverwaltung aufbauen.

3.4 Sinnvolle Hardware-Erweiterungen

Wie bereits erwähnt, benötigt Power Basic keine spezielle Hardware und ist auch auf Rechnern mit Minimalausstattung lauffähig, was für heutige Programmiersprachen bereits sehr selten der Fall ist. Aber natürlich gibt es auch sinnvolle Hardware-Erweiterungen, die die Arbeit mit Power Basic verbessern. Auch sind spezielle Fähigkeiten von Power Basic nur nutzbar, wenn eine entsprechende Rechnerausstattung vorhanden ist.

Generell kann man sagen, daß die sinnvollste Erweiterung eine Festplatte ist. Die Kapazität ist dabei nicht unbedingt von entscheidender Bedeutung für Power Basic. Auch eine 10-Mbyte-Festplatte wäre für Power Basic ausreichend. Sie sollten allerdings berücksichtigen, daß Sie sicherlich mehrere Programme auf der Festplatte installieren wollen, so daß die Kapazität nicht unter 40 Mbyte liegen sollte.

Der Rest der Hardware-Ausstattung richtet sich in erster Linie danach, welche Programme Sie vordringlich entwickeln wollen und wie wichtig Ihnen die Ablaufgeschwindigkeit der Programme ist. Wollen Sie den Schwerpunkt auf die Grafikprogrammierung legen, sollten Sie eine VGA-Grafikkarte in Ihren Rechner einbauen lassen, die bereits sehr preisgünstig zu haben ist. Um in den Genuß einer hohen Ausführungsgeschwindigkeit zu kommen, sollten Sie mindestens einen Rechner mit 80286-Hauptprozessor verwenden. Ein Coprozessor hingegen ist nur für die kommerzielle Nutzung erforderlich.

3.5 Der Kommandozeilen-Compiler

Mit Power Basic können Sie komfortabel auch größere Programme schreiben. Dennoch kann es passieren, daß nicht genügend Arbeitsspeicher für die Programmentwicklung zur Verfügung steht. In diesem Fall steht Ihnen ein Compiler ohne Entwicklungsumgebung zur Verfügung, der im Speicher selbst weniger Platz benötigt. Damit können Sie Programme über einen Befehl von der Betriebssystemebene aus in Maschinensprache übersetzen. Der Compiler befindet sich auf den Originaldisketten und hat den Namen PBC.EXE.

Zunächst wollen wir noch einmal in die Entwicklungsumgebung zurückkehren. Wenn Sie ein Programm übersetzt haben, können Sie sich über [F10] *File → Get Info* nähere Informationen zum Speicherbedarf des gerade übersetzten Programms auflisten lassen (Bild 3.5.1).

Bild 3.5.1: Das Info-Fenster von Power Basic

Ist der freie Speicher (*Free Memory*) nicht mehr sehr groß, sollten Sie mit PBC arbeiten. Aber keine Angst, in der Regel brauchen Sie den Kommandozeilen-Compiler nur in sehr seltenen Fällen. Für professionelle Programmierer aber ist er ein weiteres positives Merkmal des Power-Basic-Programmpakets.

Wenn Sie PBC direkt aufrufen, ohne eine bestimmte Quelldatei anzugeben, erhalten Sie auf dem Bildschirm eine Liste der möglichen Parameter, die Sie an den Kommandozeilen-Compiler übergeben können (Bild 3.5.2).

```
C:\SPRACHEN\PB>pbc
PowerBASIC  Version 2.10a  Copyright (c) 1990 by Robert S. Zale
Spectra Publishing, Sunnyvale, CA, USA
Syntax is: PBC [ options ] file [ options ]     * = default; -x- = turn x off
-OC        control-break test            -CE    compile to .EXE
-OA        auto array DIMension          -CU    compile to .PBU
-RE=xxxxxx find run-time error address   -CC    compile to .PBC
-ODM       generate .MAP file          * -G86   generate 8086 code
-OMS=xxxxx stack = xxxxx bytes           -G286  generate 80286 code
-OMM=xxxx  music buffer = xxxx notes    * -FEMU emulate floating point
-OMC=xxxxx com buffer = xxxxx bytes      -FNPX  NPX (80x87) float
-OMA=xx    string segment = xx Kbytes    -FP    procedural float
-EP        param error test            * -LS    com support
-EB        bounds error test           * -LP    printer support
-EO        overflow error test         * -LG    graphics support
-EN        numeric error test          * -LC    CGA graphics support
-ES        stack error test            * -LE    EGA graphics support
-DExxx     executable directory        * -LV    VGA graphics support
-DIxxx     include directories         * -LH    Hercules support
-DOxxx     object directories            -LA    all devices supported
-DUSxxx    unit source directories       -LI    interpreted print
-DUDxxx    unit destination directory    -LF    full float emulator

C:\SPRACHEN\PB>
```

Bild 3.5.2: Aufruf des Kommandozeilen-Compilers PBC

3.6 Nützliche Zusatz-Software

Power Basic ist noch nicht sehr lange auf dem Markt, so daß nur sehr wenig an Zusatzprogrammen verfügbar ist. Kirschbaum Software, also der deutsche Vertreiber von Power Basic, stellt zwei Toolboxen zur Verfügung, die die Programmierung von Programmoberflächen vereinfachen sollen. Der Preis eines Paketes beträgt ca. 90,– DM. Im Versand können die Toolboxen bereits für ca. 60,– DM bezogen werden. Entsprechende Anbieter können Sie den Computer-Fachzeitschriften entnehmen. Jede Toolbox beinhaltet ein leider nur recht knappes Handbuch. Die Funktionsbibliotheken werden sowohl auf einer 3½"- als auch einer 5¼"-Diskette ausgeliefert. Sie erhalten die fertig übersetzten Bibliotheksmodule, aber keine Quelltexte der in Assembler programmierten Routinen. Anpassungen oder Änderungen der Bibliotheksroutinen sind also nicht möglich.

Ein großer Schwachpunkt ist, daß der Inhalt der zwei Disketten sich unterscheidet, obwohl die Dateien eigentlich gleich sein sollten. Nicht jede Version ist auf jedem Rechner lauffähig. Unter Umständen müssen Sie die 5"-Diskette verwenden, um das Installationsprogramm ausführen zu können. Auch ein Systemabsturz ist bei der Installation nicht ausgeschlossen. Bevor Sie die Toolboxen einsetzen können, sind zunächst genauere Vorarbeiten notwendig, die leider im Handbuch mit keinem Wort erwähnt sind. Der Umfang der Routinen selbst macht hingegen einen besseren Eindruck.

Power Tools I

Die erste Toolbox umfaßt u.a. eine Fensterverwaltung, Menüroutinen und Ein- und Ausgabefunktionen, die die Programmierung von SAA-konformen Oberflächen erlauben. Zusätzlich stehen Hardware-nahe Funktionen sowie Datei- und Verzeichnisfunktionen zur Verfügung.

Power Tools II

Die zweite Funktionsbibliothek beinhaltet schwerpunktmäßig ein kontextsensitives Hilfesystem, das resident geladen und auch universell in eigenen Programmen genutzt werden kann. Außerdem werden Funktionen zur Mausunterstützung und Anweisungen für die freie Formatierung von Zeichenketteneingaben bereitgestellt.

Benutzt man die Power Tools I und II, lassen sich recht komfortable und professionelle Programme entwickeln. Die Einarbeitungszeit dürfte für Programmieranfänger, aufgrund der mageren Dokumentation, allerdings recht groß sein.

Kapitel 4: Referenz

In dem ersten Teil der Referenz werden wir sämtliche Menüeinträge, die in der Entwicklungsumgebung von Power Basic enthalten sind, kurz erläutern. Im zweiten Teil geben wir eine Kurzreferenz des Sprachumfanges von Power Basic. Anschließend finden Sie in der *Schnellinformation* einige Lösungen zu häufigen Programmieraufgaben der Praxis.

4.1 Menüreferenz

Jedes Menü ist nachfolgend mit dem englischen Menüeintrag aufgeführt und mit einer kurzen deutschen Erläuterung versehen. Sind für die Menüeinträge spezielle Tasten definiert, mit denen die gleiche Funktion ausgeführt werden kann, sind diese ebenfalls mit angegeben. Falls notwendig, öffnen sich nach der Auswahl eines Menüpunkts spezielle Eingabe- oder Dateiauswahlfenster, deren Bedienung wir an dieser Stelle nicht mehr genauer behandeln wollen.

4.1.1 Das Menü »File«

Im Menü *File* sind sämtliche Funktionen zur Datei- und Verzeichnisverwaltung enthalten.

Load **Laden** F3

Mit diesem Menü können Sie über ein Dateiauswahlfenster oder direkt in einem Eingabefenster den Namen der Quelldatei angeben, die in den Editor geladen werden soll.

Pick **Liste** Alt + F3

Während einer Sitzung merkt sich Power Basic die Dateinamen, die zwischendurch geladen wurden. Im Speicher ist immer nur eine einzelne Quelldatei enthalten. Wenn Sie mehrere Quelltexte wechselseitig bearbeiten müssen, können Sie einmal geladene Dateien aus dieser Liste auswählen.

New **Neu**

Mit diesem Befehl löschen Sie einen eventuell im Editor enthaltenen Text, und der Dateiname wird auf NONAME.BAS gesetzt. Sie können nun ein neues Programm eingeben.

Save **Sichern** F2

Hiermit können Sie den aktuell im Editor bearbeiteten Quelltext in eine Datei speichern. Bevor bereits vorhandene Dateien überschrieben werden, erscheint im Normalfall eine Sicherheitsabfrage.

Write to **Sichern als**

Mit diesem Befehl können Sie eine im Speicher befindliche Datei unter einem neuen Namen speichern. Sie können also eine Kopie einer Datei erzeugen. Geben Sie als Namen eine Datei an, die bereits vorhanden ist, erscheint eine Sicherheitsabfrage. Nach dem Befehl wird der Dateiname im Editor auf den neuesten Stand gebracht, so daß das nächste Speichern wieder über den Menübefehl *Save* möglich ist (Kapitel 2.2).

Directory **Verzeichnis**

Diese Menüanweisung gibt die Dateien des aktuellen Verzeichnisses in einem Dateiauswahlfenster aus. Die ausgewählte Datei wird in den Editor geladen.

Change dir **Verzeichnis wechseln**

Hiermit können Sie ein neues Arbeitsverzeichnis für Power Basic angeben. Beim Aufruf wird in einem Eingabefenster immer das aktuelle Verzeichnis angezeigt. Auch ein Laufwerkwechsel ist über diesen Menübefehl möglich.

Get Info **Informationen anzeigen**

Hiermit können Sie sich aktuelle Informationen zu Verzeichnis, Arbeitsdatei und belegtem Speicher ausgeben lassen. Insbesondere der noch zur Verfügung stehende freie Arbeitsspeicher und der Speicherbedarf für das im Speicher befindliche Programm sind dabei von Bedeutung (Kapitel 3.1).

OS Shell **Ausgang Betriebssystem**

Mit *OS Shell* können Sie temporär auf die Betriebssystem-Ebene zurückkehren, wobei die Entwicklungsumgebung und der Quelltext im Arbeitsspeicher verbleiben. Für Programme, die Sie nun unter DOS ausführen, steht nur begrenzter Speicher zur Verfügung. Bedenken Sie, daß große Programme nun nicht mehr geladen werden können. Mit EXIT gelangen Sie zurück in die Entwicklungsumgebung.

Quit **Beenden** Alt + X

Mit diesem Menü verlassen Sie Power Basic und kehren zum Betriebssystem zurück.

4.1.2 Das Menü »Edit«

Mit der Taste [E] oder Anwahl des Hauptmenüpunkts *Edit* gelangen Sie in den Editor. Um vom Editor in das Hauptmenü zurückzukehren, drücken Sie die Taste [Esc] oder [F10].

4.1.3 Das Menü »Run«

Über das Menü *Run* starten Sie Ihre im Speicher befindlichen Programme ganz oder auch nur teilweise.

Run	**Starten**	[F9]

Übersetzt und startet ein im Editor befindliches Programm.

Program reset	**Neustart**	[Strg] + [F2]

Falls Sie den Debugger verwendet haben, geht Arbeitsspeicher für die Informationen verloren, die die Programmverfolgung ermöglichen. Um diesen Speicher nach dem Debuggen wieder freizugeben, sollten Sie diesen Menüpunkt aufrufen.

Go to cursor	**Gehe bis Cursor**	[F4]

Nachdem Sie den Textcursor im Editor gezielt auf eine spezielle Anweisung positioniert haben, wird durch diese Menüanweisung das Programm genau bis an diese Stelle ausgeführt. Dieser Ausführungsmodus bietet sich in Verbindung mit dem Quelltext-Debugger an.

Trace into	**Einzelschrittmodus**	[F7]

Mit diesem Menübefehl bzw. mit der Funktionstaste [F7] können Sie Ihr Programm im Einzelschrittmodus ausführen (Kapitel 3.1). Auch benutzerdefinierte Anweisungen (Unterprogramme, Funktionen) werden in Einzelschritten abgearbeitet.

Step over **Einzelschrittmodus** F8

Diese Funktion ist nahezu identisch mit der vorangehenden. Lediglich benutzerdefinierte Funktionen und Unterprogramme werden in einem Schritt ausgeführt. Sind Sie sicher, daß ein Programmfehler nicht in einem Unterprogramm oder einer Funktion zu suchen ist, sollten Sie diese Anweisung zum Debuggen verwenden.

User screen **Ausgabemodus** Alt + F5

Mit diesem Menübefehl können Sie auf den Ausgabebildschirm umschalten. Hier sind alle Ergebnisse sichtbar, die Ihr Programm während einer Ausführung in der Oberfläche ausgibt. Die Umschaltung ist insbesondere während des Debuggen sehr wichtig.

Block run **Block ausführen**

Mit diesem Befehl haben Sie die Möglichkeit, einen Programmteil als getrennten Block zu übersetzen und auszuführen. Dazu markieren Sie im Editor den Blockanfang mit Strg + K B und das Blockende mit Strg + K K. Anschließend können Sie diesen Block mit *Block run* übersetzen und ausführen. Der Block selbst ist dann unabhängig vom übrigen Programm.

4.1.4 Das Menü »Compile«

Mit den Untermenüpunkten zu *Compile* können Sie die Codegenerierung beeinflussen, Laufzeitfehler lokalisieren bzw. eine Hauptdatei für den Compiler festlegen.

Compile **Übersetze** Alt + F9

Mit diesem Befehl können Sie ein Programm in Maschinensprache übersetzen. Dabei wird die ausführbare Programmdatei so abgelegt, wie es im nachfolgenden Menü festgelegt wird. Der Arbeitsstand der Übersetzung wird in einem Ausgabefenster angezeigt. Tritt ein Fehler auf, wird die Übersetzung beendet und in den Editor an die Fehlerstelle gesprungen. Ist das Programm fehlerfrei, bestätigt der Compiler die erfolgreiche Übersetzung. Das Programm wird nicht wie bei *Run* direkt ausgeführt.

Destination **Übersetze nach ...**

Durch diesen Menüpunkt können Sie festlegen, ob eine Quelldatei in ein ausführbares Programm, eine Unit, ein Chain-Modul oder direkt in den Speicher übersetzt werden soll.

Memory: Programm in den Speicher übersetzen.

EXE file: Ausführbare Programmdatei im aktuellen Laufwerk und im aktuellen Verzeichnis erzeugen.

Chain file: Datei ohne Runtime-Modul erzeugen. Diese Datei kann nur von einem ausführbaren Programm mit Laufzeitbibliothek über den *CHAIN*-Befehl gestartet werden.

Unit file: Funktionsbibliothek erzeugen (Kapitel 2.1).

Find Error **Suche Fehler**

Mit Hilfe dieses Menübefehls können Sie einen Laufzeitfehler, der in einem Maschinenprogramm aufgetreten ist, in dem entsprechenden Quelltext lokalisieren (Kapitel 3.1).

Main File **Hauptdatei übersetzen**

Mit dieser Anweisung können Sie eine Hauptdatei für den Compiler angeben. Jeder Übersetzungsvorgang, den Sie anschließend durchführen, wird mit dieser Datei begonnen. Diese Menüoption ist dann interessant, wenn Sie Ihr Programm in mehrere Quelldateien zerlegt haben (Kapitel 2.2).

4.1.5 Das Menü »Options«

Mit Hilfe des Optionen-Menüs können Sie Einstellungen des Compilers und Linkers festlegen und so Einfluß auf die Codegenerierung nehmen.

Compiler **Compiler konfigurieren**

Mit diesem Menübefehl nehmen Sie die Einstellungen am Compiler vor.

Generated Code: Der erzeugte Maschinencode soll entweder für den 8086/88-Prozessor oder für den 80286-Hauptprozessor erzeugt werden. Im ersten Fall ist das Programm auf jedem Rechner lauffähig, wogegen im zweiten Fall ein Rechner mit 80286-, 80386- oder 80486-Prozessor erforderlich ist.

Floating Point: Bestimmt die zu wählende Bibliothek für Fließkommaberechnungen. »Emulate« nutzt einen Coprozessor, falls vorhanden, emuliert diesen aber auch, wenn er nicht installiert ist. Mit »NPX (87)« erzeugen Sie direkt Maschinencode für einen Coprozessor. Eine Programmausführung auf einem Rechner ohne Coprozessor ist dann nicht mehr möglich. Mit »Procedure« steht Ihnen eine weitere Bibliothek für Fließkomma-Berechnungen zur Verfügung, die die Rechengeschwindigkeit auch ohne Coprozessor beschleunigt.

Array auto-dim: »On« sorgt dafür, daß Datenfelder automatisch auf zehn Elemente dimensioniert werden. »Off« schaltet diese Dimensionierung, die auch für andere Basic-Dialekte typisch ist, aus. In diesem Fall muß jedes Datenfeld explizit dimensioniert werden.

Control-Break testing: Schalten Sie dieses Menü auf »On«, kann Ihr Programm während Ein- und Ausgaben mit der Tastenkombination [Strg] + [Break] bzw. [Strg] + [Untbr] abgebrochen werden. Bei »Off« ist ein vorzeitiger Programmabbruch und damit die Rückkehr zum Betriebssystem nicht möglich.

Error tests: Mit diesem Menü haben Sie die Möglichkeit, speziellen Code für die Programmüberwachung zu erzeugen. Mit Hilfe dieser Testroutinen können Sie während der Programmentwicklung Systemabstürze vermeiden. Dazu sollten Sie sämtliche Error-Schalter auf »On« setzen. Zur Reduzierung der Code-Größe können Sie ein ausgetestetes Programm je nach Bedarf auch ohne diese Fehlertests übersetzen lassen.

Stack Test On: Überprüfung, ob genügend Stapelspeicher für lokale Variablen und Parameterwerte von Unterprogrammen und Funktionen vorhanden ist.

Bounds test On: Überprüfung, ob die Grenzen von Datenfeldern eingehalten werden.

Overflow test On: Überprüfung von Ganzzahlen- und Fließkommazahlen-Datentypen auf Bereichsüberschreitung.

Numeric test On: Überprüfung, ob Rechenfehler entstehen, wie z.B. Division durch Null.

Param test On: Überprüfung auf interne Fehler. Sollte standardmäßig auf »On« gesetzt sein.

Metastatements: Mit diesem Menü können spezielle Speicherbereiche in ihrer Größe festgelegt werden.

Stack Size: Änderung der Größe des Stapelspeichers (von standardmäßig 1536 auf maximal 32766 Byte).

Music Buffer: Pufferspeicher für Hintergrundmusik festlegen (0 bis maximal 8 Noten, wobei jede Note 8 Byte Speicher belegt).

Com Buffer: Pufferspeicher für seriellen Datenaustausch über COM1 und COM2 festlegen (0 bis maximal 32767 Byte).

Alloc string seg: Größe neu belegter Zeichenkettensegmente festlegen (voreingestellt 32, was 32750 Byte entspricht). Durch die Reduzierung dieses Werts können Sie mehr Zeichenketten in Ihrem Programm benutzen, die maximale Länge einer Zeichenkette wird dann allerdings reduziert. Sie können die Werte 1, 2 ,4, 8, 16 und 32 als Segmentgröße festlegen, was einer jeweiligen Stringlänge von 1006, 2030, 4078, 8174, 16366 bzw. 32750 Zeichen entspricht.

Linker

Über das Linker-Menü können Sie explizit festlegen, welche Module aus der Power-Basic-Laufzeitbibliothek in ein Programm mit eingebunden werden sollen. Damit läßt sich die Programmgröße eines eigenständig ablauffähigen Maschinenprogramms reduzieren.

Seriell Communications off: Keine Routine für die serielle Schnittstelle in das Programm einbinden.

Printer Off Keine Anweisungen zur Druckerunterstützung einbinden (parallele Schnittstelle).

Full float emulation Off: In diesem Fall werden nur die benötigten mathematischen Funktionen in Ihr Programm eingebunden. Schalten Sie dieses Menü auf »On«, so wird die komplette Bibliothek zur Emulation der Fließkommaberechnungen mit eingebunden.

Interpreted Print Off: Sämtliche ASCII-Zeichen (auch ASCII 0 bis ASCII 32) werden als darstellbare Zeichen ausgegeben. Steht der Menüschalter auf »On«, werden die ASCII-Zeichen von 0 bis 32 als Steuerungscodes interpretiert.

Graphics Off: In Ihr Programm werden keine Grafikroutinen mit eingebunden.

Video Cards: Hiermit können Sie wahlfrei festlegen, welche Grafikkarten Ihr Programm unterstützen soll. Unterstützt Ihr Programm lediglich die VGA-Karte, wie z.B. auch unser Programm HAUSHALT, so schalten Sie VGA auf »On« und alle übrigen Grafikkarten auf »Off« (Kapitel 2.10). Die Grafikbibliothek beinhaltet Routinen für die CGA-, EGA-, VGA- und Hercules-Grafikkarte. Werden obengenannte Teilmodule in einem Programm benötigt, sind diese über den Schalter »On« in das Programm einzubinden.

Environment Umgebung

Über diesen Menüpunkt können Sie die Power-Basic-Umgebung an Ihre eigenen Bedürfnisse anpassen.

Config auto save On: Schalten Sie diesen Menüpunkt mit »On« ein, werden geänderte Konfigurationen automatisch gespeichert und auch beim nächsten Programmstart von Power Basic wieder aktiv. Dazu gehören z.B. die Optionen, die Sie für den Compiler und Linker festgelegt haben.

Edit auto save On: Sorgt für eine automatische Zwischenspeicherung einer geänderten Quelldatei vor jedem *Step over, Run* oder *OS Shell.*

Warn On Overwrite On: Speichern Sie Dateien, die bereits auf dem Festspeicher vorhanden sind, erscheint eine Sicherheitsabfrage, ob Sie diese Datei überschreiben wollen. Beachten Sie, wenn Sie diese Option auf »Off« schalten, werden Dateien bei Namensgleichheit ohne Vorwarnung überschrieben!

Backup Files On: Beim Speichern legt Power Basic von Ihrer Quelldatei automatisch eine Sicherungskopie mit dem Dateikürzel .BAK an. Diese wird im selben Verzeichnis abgelegt, in der sich Ihre Quelldatei befindet. Mit der Option »Off« können Sie das Erzeugen einer Sicherungskopie unterdrücken. Sie sollten regelmäßig Ihre Quelldatei zusätzlich auf Diskette speichern.

Tab Size: Über diesen Menüpunkt kann die Sprungweite für die Tabulatortaste eingestellt werden (2 bis maximal 16 Leerzeichen). Beachten Sie, daß bei der Verwendung von Tabulatoren standardmäßig Tab-Zeichen (ASCII 15) in den Quelltext geschrieben werden, die nicht von jedem Editor erkannt werden. Um statt dieser Tabulatoren Leerzeichen in den Quelltext zu schreiben, müssen Sie die Tastenkombination [Strg] + [O] + [T] betätigen. Am oberen Rand des Editor-Fensters wird angezeigt, ob Sie mit oder ohne Tabulatoren arbeiten.

Zoom Windows On: Mit diesem Menüpunkt können Sie das jeweils aktuelle Edit-, Watch- bzw. Output-Fenster auf Bildschirmgröße zoomen. »Off« stellt das Edit- und Watch-Fenster parallel auf dem Bildschirm dar.

Screen Size: Unter der Voraussetzung, daß Sie über die entsprechende Hardware verfügen, können Sie die Anzahl der parallel auf dem Bildschirm dargestellten Zeilen erhöhen. Standardmäßig werden 25 Zeilen auf dem Bildschirm dargestellt, bei einer EGA-Karte können jedoch auch 43 und bei einer VGA-Karte sogar 50 Zeilen ausgegeben werden.

Directories Verzeichnisse

Unter diesem Punkt können Sie spezielle Verzeichnisse für bestimmte Dateien von Power Basic festlegen.

Power Basic directory: Festlegung des Suchpfades für die Power-Basic-Hilfs- und Konfigurationsdatei. Diese Dateien werden in Verbindung mit der Entwicklungsumgebung benötigt.

Executable directory: Angabe eines Verzeichnisses, in das die vom Compiler erstellte Programmdatei gespeichert werden soll. Ohne eine Angabe wird die EXE-Datei im aktuellen Verzeichnis gespeichert.

Include directories: Suchpfadfestlegung der Einfügedateien. Mehrere Pfade können durch ein Semikolon getrennt eingegeben werden. In diesen Verzeichnissen sucht der Compiler später nach benötigten Include-Dateien. Ohne eine Angabe von Verzeichnissen wird nur im aktuellen Verzeichnis gesucht.

Unit Source directories: Auch für die Quelldateien der Units (Funktionsbibliotheken) können mehrere Pfade, durch Semikolon getrennt, eingegeben werden.

Unit Destination directory: In diesem Verzeichnis werden die übersetzten Units (PBU-Dateien) abgespeichert. Geben Sie hier keinen Pfad an, werden die Units im aktuellen Verzeichnis gespeichert. Benötigt der Compiler später eine Unit, sucht er zunächst im aktuellen und anschließend in dem hier angegebenen Verzeichnis

Object directories: Hier können Sie ein Verzeichnis angeben, in dem Sie die Module eines Assemblers speichern (OBJ- und COM-Dateien), die später in ein Power-Basic-Programm integriert werden sollen. Benötigt der Compiler ein Assembler-Modul, so sucht er zunächst im aktuellen und anschließend in dem hier angegebenen Verzeichnis.

Pick File Name: Unter diesem Menüpunkt können Sie das Verzeichnis und den Dateinamen für die Pick-Liste festlegen (standardmäßig PBPICK.PBP). In dieser Datei werden später die Dateinamen gespeichert, die Sie über den Menüpunkt *File → Pick* bzw. mit [Alt] + [F3] zur Auswahl angezeigt bekommen.

Current Pick File: Unter diesem Menüpunkt erhalten Sie die aktuellen Informationen zu Pfad und Dateinamen der aktuellen Pick-Datei.

Parameters **Parameter**

Unter diesem Menüpunkt können Sie innerhalb der Entwicklungsumgebung einen Kommandostring definieren, der von einem Programm, innerhalb der Oberfläche von Power Basic, über die Funktion *COMMAND$* genutzt werden kann.

Save Options **Optionen sichern**

Speicherung aller gewählten Optionen für Compiler, Linker und Environment in eine Konfigurationsdatei.

Retrieve Options **Optionen holen**

Laden einer Konfigurationsdatei. Sie können mit mehreren Konfigurationsdateien arbeiten, die Sie je nach Bedarf und Projekt laden können.

4.1.6 Das Menü »Debug«

In diesem Menü sind wichtige Befehle für den Debugger zusammengefaßt, die die Fehlersuche in Programmen vereinfachen sollen.

Evaluate **Auswertung** [Strg] + [F4]

Über diesen Menüpunkt können Sie während des Debuggens Variablenwerte ändern (Kapitel 3.1) und somit die Programmausführung beeinflussen.

Call stack **Programmlauf zeigen** Strg + F3

Über diesen Menüpunkt können Sie sich während des Debuggens Informationen darüber ausgeben lassen, wie Sie an die derzeit aktuelle Programmposition gelangt sind. Sie können sehen, in welcher Reihenfolge Funktionen und Unterprogramme aufgerufen wurden.

Find SUB/function/ label **Unterprogramm/Funktion/Marke suchen**

Nachdem Sie Ihr Programm kompiliert haben, können Sie über diesen Menüpunkt gezielt nach Namen von Unterprogrammen, Funktionen und Sprungmarken suchen lassen. Dabei werden auch zu einem Hauptprogramm gehörende Einfügedateien (Include-Dateien) und Funktionsbibliotheken (Units) berücksichtigt.

Integrated Debugging **Fehlersuche**

Mit »On« schalten Sie den Debugger-Modus ein und mit »Off« aus. Innerhalb der Oberfläche steht Ihnen geringfügig mehr Speicher für das auszuführende Programm zur Verfügung, wenn Sie den Debugger ausschalten.

MAP File **MAP-Datei**

Hiermit können Sie den Debugger anweisen, eine spezielle MAP-Datei zu erzeugen, wenn Sie eine ausführbare EXE-Datei erstellen (*Compile → Destination → EXE file*). Dann haben Sie die Möglichkeit, einen externen Debugger zu benutzen, um ein Power-Basic-Programm nach Fehlern zu durchsuchen. Verfügen Sie nicht über einen externen Debugger, sollten Sie diese Option auf »Off« stellen, da sonst die Größe der erzeugten Programmdateien durch zusätzliche Debugger-Informationen unnötig erhöht wird.

Display swapping **Bildschirmumschaltung**

Über diesen Menüpunkt können Sie festlegen, ob und auf welche Weise während der Debugger-Phase der Bildschirm umgeschaltet wird. Zum einen muß ja die Entwicklungsumgebung angezeigt werden, und zum

anderen müssen auch die Ausgaben durch das Programm selbst dargestellt werden. Bei »Smart« wird immer dann zu Ihrem Programmbildschirm umgeschaltet, sobald Ihr Programm eine Bildschirmausgabe vornimmt. »Always« hingegen schaltet nach jeder Anweisung den Bildschirm um. Diese Methode ist besonders sinnvoll, wenn Sie den Bildschirmspeicher von Power Basic direkt manipulieren. Mit »None« können Sie die Umschaltung zwischen den Bildschirmen unterdrücken.

Refresh Display **Bildschirmauffrischung**

Mit diesem Menüpunkt können Sie den Bildschirmaufbau der integrierten Entwicklungsumgebung wieder korrigieren, falls Ihr Programm den Bildschirm zum Teil überschrieben hat.

4.1.7 Das Menü »Break/Watch«

Dieser Menüpunkt enthält die wichtigsten Anweisungen, die Sie in Verbindung mit dem Debugger benötigen. Über dieses Menü können Sie Variablen verfolgen und Haltepunkte setzen (Kapitel 3.1).

Add watch **Variable verfolgen** Strg + F7

Über diesen Menüpunkt können Sie eine Variable eingeben, deren Werte Sie im Watch-Fenster verfolgen wollen. Auch Ausdrücke lassen sich in Power Basic verfolgen.

Delete watch **Variable löschen**

Über diesen Menüpunkt können Sie den Befehl zur Verfolgung einer Variablen wieder rückgängig machen.

Edit watch **Variable bearbeiten**

Haben Sie die zu verfolgende Variable falsch eingegeben, können Sie über diesen Menüpunkt eine Korrektur vornehmen.

Remove all watches	**alle Variablen löschen**	

Mit dieser Menüanweisung können Sie sämtliche Variablen aus dem Watch-Fenster wieder entfernen.

Toggle breakpoint	**Haltepunkt setzen**	Strg + F8

Über diesen Menüpunkt können Sie Haltepunkte setzen bzw. wieder löschen. Positionieren Sie dazu den Textcursor im Editor genau in die Quelltextzeile, an der der Haltepunkt eingefügt oder gelöscht werden soll.

Clear all breakpoints	**Haltepunkt löschen**

Mit dieser Menüanweisung löschen Sie sämtliche Haltepunkte, die Sie in dem Programm gesetzt haben.

View next breakpoint	**nächsten Haltepunkt zeigen**

Mit diesem Menüpunkt springen Sie im Editor zum jeweils nächsten Haltepunkt. Ist der letzte Haltepunkt im Programm erreicht, wird wieder mit dem ersten begonnen.

4.2 Sprachreferenz

Anschließend finden Sie sämtliche Befehle, die Sie in Power Basic verwenden können, mit einer kurzen Erläuterung und einer Kurzfassung der Syntax. Genauere Informationen können der Hilfefunktion und den Handbüchern von Power Basic entnommen werden. Sämtliche Anweisungen sind speziellen Themenkreisen zugewiesen. Wenn mehrere Befehle in mehrere Teilbereiche eingruppierbar sind, werden die Funktionen dem häufigeren Einsatzgebiet zugeordnet. Optionale Parameter sind in eckige Klammern eingefaßt, Alternativmöglichkeiten werden durch »|« getrennt. Einige Anweisungen wurden zur Veranschaulichung vereinfacht.

4.2.1 Compiler-Anweisungen

$COM

Speicherplatz für den Empfangspuffer der seriellen Schnittstellen reservieren. Diese Festlegung bezieht sich auf alle vier der von Power Basic unterstützten Ports.

```
$COM Größe in Byte
```

$COMPILE

Legt fest, ob ein Quelltext in den Speicher, in eine ausführbare Programmdatei, ein *CHAIN*-Modul oder eine Funktionsbibliothek übersetzt werden soll.

```
$COMPILE MEMORY|EXE|CHAIN|UNIT
```

$CPU

Legt fest, für welchen Prozessor das Programm erzeugt wird.

```
$CPU 8086|80286
```

$DEBUG

Legt fest, ob eine MAP-Datei für einen externen Debugger erzeugt werden soll.

```
$DEBUG MAP [ON|+|OFF|-]
```

$DYNAMIC

Datenfelder werden standardmäßig als dynamische Datenfelder verwaltet. Die Größe der Datenfelder wird während des Programmlaufs festgelegt.

```
$DYNAMIC
```

$ERROR

Legt fest, welche Fehlerkontrollcodes in ein Programm eingebunden werden sollen.

```
$ERROR Test [Status],...
       Test = BOUNDS, NUMERIC, OVERFLOW, PARAM, STACK, ALL
       Status= ON, +, OFF, -
```

$EVENT

Legt fest, ob Programmcode für die Ereignisverfolgung erzeugt wird oder nicht.

```
$EVENT ON|+|OFF|-
```

$FLOAT

Festlegung der Fließkommabibliothek.

```
$FLOAT EMULATE|NPX|PROCEDURE
```

$IF/$ELSE/$ENDIF

Programmteile werden nur übersetzt, wenn eine bestimmte Konstante definiert ist (konditionale Konditionierung).

```
$IF Konstante
    'Anweisungen
  [$ELSE
    'Anweisungen]
$ENDIF
```

$INCLUDE

Weist den Compiler an, eine Quelldatei einzubinden.

```
$INCLUDE Dateiname
```

$INLINE

Weist den Compiler an, Maschinencode direkt in ein Programm einzufügen.

```
$INLINE Dateiname|Bytefolge
```

$LIB

Legt fest, welche Module der Laufzeitbibliothek in das ausführbare Maschinenprogramm mit eingebunden werden.

```
$LIB Bibliothek [Status]
     Bibliothek: COM, LPT, GRAPH, CGA, EGA, VGA, HERC,
                 ULLFLOAT, IPRINT, ALL
     Status:     ON, +, OFF, -
```

$LINK

Weist den Compiler an, eine Objekt- oder Unit-Datei in das Programm mit einzubinden.

```
$LINK Dateiname
```

$OPTION

Festlegung von Compiler-Optionen.

```
$OPTION Option [Status],...
        Option: AUTODIM, CNTBREAK
        Status: ON, +, OFF, -
```

$SEGMENT

Deklariert ein neues Code-Segment.

```
$SEGMENT
```

$SOUND

Legt die Puffergröße für Hintergrundmusik fest.

```
$SOUND Größe in Byte
```

$STACK

Festlegung des Stapelspeichers.

```
$STACK Größe in Byte
```

$STATIC

Datenfelder werden standardmäßig als statisch deklariert, d.h., die Größe des Datenfeldes ist bereits während der Übersetzung festgelegt.

```
$STATIC Größe in Byte
```

$STRING

Legt die Segmentgröße für Zeichenketten fest.

```
$STRING 1|2|4|8|16|32
```

4.2.2 Compiler-Daten und Variablendeklarationen

DATA

Konstantendefinition, die später durch *READ*-Befehle gelesen werden.

```
DATA Konstante,...
```

DEFBCD

Variablen als BCD-Fließkommazahlen deklarieren.

```
DEFBCD Buchstabenbereich
```

DEFDBL

Variablen als Fließkommazahlen mit doppelter Genauigkeit deklarieren.

```
DEFDBL Buchstabenbereich
```

DEFEXT

Variablen als Zahlen mit erweiterter Genauigkeit deklarieren.

```
DEFEXT Buchstabenbereich
```

DEFFLX

Variablen als Flex-String deklarieren.

```
DEFFLX Buchstabenbereich
```

DEFINT

Variablen als Ganzzahlen deklarieren.

```
DEFINT Buchstabenbereich
```

DEFLNG

Variablen als lange Ganzzahlen deklarieren.

```
DEFLNG Buchstabenbereich
```

DEFQUD

Variablen als Quad-Integer deklarieren.

```
DEFQUD Buchstabenbereich
```

DEFSNG

Variablen als Fließkommazahlen mit einfacher Genauigkeit deklarieren.

```
DEFSNG Buchstabenbereich
```

DEFSTR

Variablen als String deklarieren.

```
DEFSTR Buchstabenbereich
```

EXTERNAL

Variablen aus einem Hauptprogramm innerhalb einer Unit als extern deklarieren (*PUBLIC*).

```
EXTERNAL Variable,...
```

LOCAL

Variablen innerhalb einer Funktion oder eines Unterprogramms als lokal deklarieren. Die Variable ist im Hauptprogramm dann nicht bekannt.

```
LOCAL Variable,...
```

PUBLIC

Variablen in einem Hauptprogramm für eine Unit oder Objektdatei als öffentlich deklarieren (*EXTERNAL*).

SHARED

Variablen innerhalb eines Hauptprogramms als global deklarieren.

```
SHARED Variable,...
```

STATIC

Variablen innerhalb eines Unterprogramms oder einer Funktion als statisch deklarieren. Statische Variablen werden nur am Programmanfang initialisiert und anders als lokale Variablen nicht bei jedem Unterprogramm- bzw. Funktionsaufruf wieder auf den Wert Null bzw. eine Nullzeichenkette gesetzt.

```
STATIC Variable,...
```

SWAP

Vertauschung von zwei Variablenwerten.

```
SWAP Variable1, Variable2
```

4.2.3 Dateien

CLOSE

Befehl zum Schließen von Dateien.

```
CLOSE #Dateinummer,...
```

EOF

Dateiende-Status abfragen. Diese Funktion gibt einen Wert ungleich Null zurück, wenn das Dateiende erreicht ist.

```
y=EOF(Dateinummer)
```

FIELD

Datenfeld-Definition im Dateipuffer für Direktzugriffsdateien.

```
FIELD #Dateinummer, Größe AS Stringvariable,...
```

FILEATTR

Dateiinformation abfragen (1=Input, 2=Output, 4=Random, 8=Append, 16=Binär).

```
y=FILEATTR(Dateinummer,1)
```

FREEFILE

Freie Dateinummer ermitteln.

```
y=FREEFILE
```

GET

Datensatz einlesen.

```
GET #Dateinummer, Datensatznummer
```

GET$

Zeichenkette aus einer binären Datei lesen.

```
GET$ #Dateinummer, AnzahlZeichen, Stringvariable
```

INPUT#

Daten aus einer sequentiellen Datei lesen und Variablen zuweisen.

```
INPUT #Dateinummer, Variable,...
```

LINE INPUT#

Eine Zeile aus einer sequentiellen Datei einlesen.

```
LINE INPUT #Dateinummer, Stringvariable
```

LOC

Position innerhalb einer Datei zurückliefern.

```
y=LOC(Dateinummer)
```

LOCK

Zugriff im Netzwerk für bestimmte Datensätze sperren.

```
LOCK #Dateinummer, Datensatz
LOCK #Dateinummer, ErsterDatensatz TO LetzterDatensatz
```

LOF

Dateigröße in Byte ermitteln.

```
y=LOF(Dateinummer)
```

LSET

Feldvariablen linksbündig in den Puffer einer Direktzugriffsdatei eintragen.

```
LSET FeldVariable=Zeichenkette
```

OPEN

Datei öffnen.

```
OPEN Datei [FOR Modus] [ACESS Zugriff] [sperren] AS #Nummer [LEN=Länge]
     Modus:   OUTPUT, INPUT, APPEND, RANDOM, BINARY
     Zugriff: READ, WRITE, READ WRITE
     Sperren: SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE
```

PRINT#

Werte in eine sequentielle Zeile schreiben.

```
PRINT #Dateinummer, Variable,...
```

PRINT# USING

Werte formatgebunden in eine sequentielle Datei schreiben.

```
PRINT #Dateinummer, USING Formatstring; Folge von Ausdrücken [;]
```

PUT

Datensatz in eine Direktzugriffsdatei schreiben.

```
PUT #Dateinummer[, Datensatznummer]
```

PUT$

Zeichenkette in eine binäre Datei schreiben.

```
PUT #Dateinummer, Zeichenkette
```

RESET

Dateipuffer auf Festspeicher sichern und alle offenen Dateien bzw. Peripheriegeräte schließen.

```
RESET
```

RSET

Feldvariablen rechtsbündig in den Puffer einer Direktzugriffsdatei eintragen.

```
RSET FeldVariable=Zeichenkette
```

SEEK

Funktion, um die nächste Datensatznummer in einer Direktzugriffsdatei bzw. die Byte-Position in einer sequentiellen oder Binärdatei zu ermitteln.

```
s=SEEK(Dateinummer)
```

SEEK

Befehl, um den Zeiger auf eine bestimmte Position innerhalb einer Random- oder Binärdatei zu setzen.

```
SEEK #Dateinummer, Position
```

UNLOCK

Aufhebung von Dateizugriffsbeschränkungen.

```
UNLOCK #Dateinummer, Datensatz
UNLOCK #Dateinummer, ErsterDatensatz TO LetzterDatensatz
```

WRITE#

Schreiben von Werten in eine sequentielle Datei. Gegenüber *PRINT#* dürfen die Ausdrücke auch Feldtrenner, z.B. Kommata, enthalten.

```
WRITE #Dateinummer, Liste von Ausdrücken
```

4.2.4 Datenfeldoperationen

ARRAY SORT

Beliebiges Datenfeld sortieren (hier vereinfacht).

```
ARRAY SORT Feld ([index]) [FOR zahl][,ASCEND|DESCEND]
           ASCEND =aufsteigende Sortierung
           DESCEND=absteigende Sortierung
```

ARRAY SCAN

Datenfeld nach einem bestimmten Element durchsuchen (hier vereinfacht).

```
ARRAY SCAN Feld ([index]) [FOR zahl], Operator, TO iVar
```

ARRAY INSERT

Ein Element in ein Datenfeld einfügen.

```
ARRAY INSERT Feld ([index]) [FOR zahl] [,Ausdruck]
```

ARRAY DELETE

Ein Element aus einem Datenfeld löschen.

```
ARRAY DELETE Feld ([index]) [FOR zahl] [,Ausdruck]
```

DIM

Anweisung zum Dimensionieren von Datenfeldern. Sowohl die obere Feldgrenze als auch die untere ist frei festlegbar. Ferner kann bereits bei der Dimensionierung festgelegt werden, ob ein Datenfeld statisch oder dynamisch sein soll.

```
DIM [STATIC|DYNAMIC] Feldname(UntereGrenze:ObereGrenze),...
```

ERASE

Löschen dynamischer und Zurücksetzen statischer Datenfelder.

```
ERASE Datenfeldname,...
```

LBOUND

Untere Feldgrenze eines Datenfeldes ermitteln.

```
y=LBOUND(Datenfeld(Dimension))
```

MAP

Definition dynamischer Datenstrukturen bzw. Variablendefinition einer Direktzugriffsdatei (hier vereinfacht).

Mit Flex-String:

```
MAP Haupt$$ *Länge, TeilLänge AS Unter$$,...
```

Mit Direktzugriffsdatei:

```
MAP #Dateinummer, Länge AS Variable$$
```

OPTION BASE

Untere Datenfeldgrenze festlegen. Diese Grenze darf zwischen 0 und 32767 liegen. Für Binärdateien können Sie die untere Grenze, also die erste Byte-Position, entweder auf 0 oder 1 setzen.

```
OPTION [ARRAY|BINARY] BASE UntereGrenze
```

REDIM

Neudimensionierung bzw. Löschen von dynamischen Datenfeldern.

```
REDIM Datenfeld(UntereGrenze:ObereGrenze),...
```

UBOUND

Obere Feldgrenze eines Datenfeldes ermitteln.

```
y=UBOUND(Datenfeld(Dimension))
```

4.2.5 DOS-Anweisungen

ATTRIB

Funktion zum Ermitteln eines DOS-Dateiattributes.

```
y=ATTRIB(Dateiname)
```

ATTRIB

Befehl zum Setzen eines DOS-Dateiattributes (attribute: 1=nur lese, 2=versteckt, 4=Systemdatei, 32=Archiv).

```
ATTRIB Dateiname, Attribut
```

CHDIR

Aktuelles Verzeichnis wechseln.

```
CHDIR Pfad
```

CHDRIVE

Aktuelles Laufwerk wechseln (A, B, C usw.).

```
CHDRIVE Laufwerk
```

COMMAND$

Aufrufparameter ermitteln, die beim Programmstart verwendet wurden.

```
a$=COMMAND$
```

CURDIR$

Aktuelles Verzeichnis auf einem beliebigen Laufwerk ermitteln.

```
a$=CURDIR$(Laufwerk)
```

DATE$

Systemdatum ermitteln.

```
a$=DATE$
```

DATE$

Systemdatum setzen.

```
DATE$= a$
```

DIR$

Dateinamen einer entsprechenden Maske ermitteln (z.B. *.BAS oder *.*). Der erste Aufruf legt die Maske fest und ermittelt den ersten passenden Dateieintrag. Nachfolgende Aufrufe in der Form *a$=DIR$* geben jeweils den nächsten passenden Eintrag zurück (Kapitel 2.2).

```
a$=DIR$ [(Maske[,Attribut])]
```

ENVIRON

DOS-Umgebungsbereich ändern (sinnvoll in Verbindung mit *SHELL*-Befehl).

```
ENVIRON Umgebungszeichenkette
```

ENVIRON$

Informationen aus der DOS-Umgebung ermitteln.

```
a$=ENVIRON$(Parameterzeichenkette|Eintragsnummer)
```

FILES

Aktuelles Verzeichnis auf dem Bildschirm anzeigen.

```
FILES [Maske]
```

KILL

Datei(en) von Diskette oder Festplatte löschen.

```
KILL Dateiname|Maske
```

MKDIR

Unterverzeichnis anlegen.

```
MKDIR Pfadname
```

NAME

Datei umbenennen.

```
NAME AlterName AS NeuerName
```

RMDIR

Unterverzeichnis löschen. Beachten Sie, daß ein Unterverzeichnis nur gelöscht werden kann, wenn sich darin keine Dateien befinden!

```
RMDIR Pfadname
```

TIME$

Funktion zum Ermitteln der Systemzeit.

```
a$=TIME$
```

TIME$

Befehl zum Setzen der Systemzeit.

TIME$=a$

4.2.6 Druckersteuerung

LPOS

Anzahl der Zeichen in der aktuellen Zeile des Druckerpuffers ermitteln (Drucker: 0 oder 1=LPT1, 2=LPT2, 3=LPT3).

```
y=LPOS(Drucker)
```

LPRINT

Daten zum Drucker senden.

```
LPRINT Variable,...[;]
```

LPRINT USING

Daten formatiert zum Drucker schicken.

```
LPRINT USING Formatzeichenkette; Variable [;]
```

SPC

Bestimmte Anzahl von Leerzeichen ausgeben (nur in Verbindung mit LPRINT, PRINT oder PRINT# verwendbar).

```
SPC(Anzahl)
```

TAB

Bestimmte Anzahl von Leerzeichen ausgeben (nur in Verbindung mit LPRINT, PRINT oder PRINT# verwendbar).

```
TAB(Anzahl)
```

WIDTH

Zeilenbreite des Bildschirms, Druckers oder eines Kommunikations-Ports festlegen (Gerät: LPT1...LPT3, COM1...COM4, SCRN).

```
WIDTH [Gerät|#Dateinummer,] Größe
```

4.2.7 Eingabe- und Ausgabeanweisungen

INKEY$

Zeichen aus dem Tastaturpuffer auslesen, ohne es auf dem Bildschirm auszugeben.

```
a$=INKEY$
```

INPUT

Eingabe anfordern und einer Variablen zuweisen.

```
INPUT [Vorgabezeichenkette;] Variable,...
```

INPUT$

Bestimmte Anzahl von Zeichen von der Tastatur oder aus einer Datei einlesen.

```
a$=INPUT$(Anzahl)[,#Dateinummer]
```

INSTAT

Status des Tastaturpuffers ermitteln. Der Wert, den diese Funktion zurückgibt, wird zu Null, sobald ein Zeichen im Tastaturpuffer enthalten ist. Ein Auslesen des Tastaturpuffers erfolgt durch diese Funktion nicht.

```
y=INSTAT
```

LINE INPUT

Eine Zeile über die Tastatur einlesen, wobei Feldtrenner, wie z.B. Kommata, ignoriert werden.

```
LINE INPUT [Vorgabezeichenkette;] Variable,...
```

PRINT

Informationen auf dem Bildschirm ausgeben.

```
PRINT Variable,...[;]
```

PRINT USING

Informationen formatgebunden auf dem Bildschirm ausgeben.

```
PRINT USING Formatzeichenkette; Variable,...
```

READ

Die Konstanten aus *DATA*-Zeilen in Variablen einlesen.

```
READ Variable,...
```

WRITE

Werte, getrennt durch Kommata, auf dem Bildschirm ausgeben.

```
WRITE Variable,...
```

4.2.8 Ereignis- und Hardware-Verfolgung

COM(n)

Eingabe über serielle Schnittstelle erlauben bzw. unterbinden.

```
COM 1|2|3|4 [ON|OFF|STOP]
```

DELAY

Programm für eine bestimmte Anzahl von Sekunden unterbrechen.

```
DELAY Sekunden
```

INP

Ein Byte von einem I/O-Port lesen.

```
y=INP(Port-Nummer)
```

IOCTL

Kontrollstring an einen Gerätetreiber schicken.

```
IOCTL #Dateinummer, Kontrollstring
```

IOCTL$

Informationen von einem Treiber empfangen.

```
s$=IOCT$(#Dateinummer)
```

KEY

Mit dieser Anweisung können die Definitionen von Funktionstasten vorgenommen bzw. am unteren Bildschirmrand ein- bzw. ausgeschaltet werden.

```
KEY ON|OFF|LIST
KEY Nummer, Zeichenkette
KEY Nummer, CHR$(Shiftstatus, Scancode)
```

KEY(n)

Ein- bzw. Ausschalten der Überwachung für eine bestimmte Taste.

```
KEY(Nummer) ON|OFF|STOP
```

MTIMER

Funktion zum Lesen des Mikrosekunden-Zählers.

```
y=MTIMER
```

MTIMER

Befehl zum Setzen des Mikrosekunden-Zählers.

```
MTIMER
```

ON COM(n)

Unterprogramm mit Sprungmarke festlegen, welches die eingehenden Daten der seriellen Schnittstellen empfängt.

```
ON COM(1|2|3|4) GOSUB Sprungmarke
```

ON KEY(N)

Sprungmarke festlegen, die beim Tastendruck einer bestimmten Taste angesprungen werden soll.

```
ON KEY(Nummer) GOSUB Sprungmarke
```

ON PEN

Sprungmarke des Unterprogramms festlegen, welches bei der Statusänderung eines angeschlossenen Lichtgriffels angesprungen werden soll.

```
ON PEN GOSUB Sprungmarke
```

ON PLAY

Unterprogramm festlegen, welches angesprungen wird, wenn der Musikpuffer nur noch eine bestimmte Anzahl von Noten enthält.

```
ON PLAY (Notenzahl) GOSUB Sprungmarke
```

ON STRIG

Unterprogramm festlegen, welches ausgeführt werden soll, wenn der Knopf eines angeschlossenen Joysticks betätigt wird.

```
ON STRIG (ButtonNummer) GOSUB Sprungmarke
```

ON TIMER

Unterprogramm festlegen, welches alle n Sekunden aufgerufen werden soll.

```
ON TIMER(n) GOSUB Sprungmarke
```

OPEN

Peripheriegerät im Dateimodus öffnen.

```
OPEN Name [FOR MODUS] AS #Dateinummer
     Name: KYBD     - Tastatur
           SCRN     - Bildschirm
           LPT1-3   - 1., 2. oder 3.parallele Schnittstelle
           COM1-4   - 1., 2., 3. oder 4. serielle Schnittstelle
```

OUT

Wert direkt auf I/O-Port ausgeben.

```
OUT Port-Nummer, Byte
```

PEN

Status des Lichtgriffels einlesen.

```
y=PEN(Option)
```

PEN

Befehl zum Ein- bzw. Ausschalten des Lichtgriffels.

```
PLAY ON|OFF|STOP
```

PLAY

Funktion zum Ermitteln der Notenanzahl im Musikpuffer.

```
y=PLAY(x)
```

PLAY

Befehl zum Erzeugen von Musik.

```
PLAY ON|OFF|STOP
PLAY Zeichenkette
```

STICK

Stellung eines angeschlossenen Joysticks zurückliefern.

```
y=STICK(Option)
```

STRIG

Status der Feuerknöpfe eines angeschlossenen Joysticks ermitteln.

```
y=STRIG(Option)
```

STRIG

Befehl zum Ein- bzw. Ausschalten der Joystick-Feuerknöpfe.

```
STRIG ON|OFF
STRIG(n) ON|OFF
```

TIMER

Anzahl der Sekunden ermitteln, die seit Mitternacht vergangen sind.

```
y=TIMER
```

TIMER

Befehl zur Überwachung der *TIMER*-Ereignisse.

```
TIMER ON|OFF|STOP
```

4.2.9 Fehlerbehandlung

ERADR

Fehleradresse des zuletzt aufgetretenen Fehlers ermitteln.

```
y&=ERADR
```

ERDEV/ERDEV$

Fehlerstatus eines Peripheriegerätes bzw. eines Treiberprogramms ermitteln.

```
y=ERDEV
y$=ERDEV$
```

ERL

Zeilennummer des zuletzt aufgetretenen Fehlers im Programm ermitteln.

```
y=ERL
```

ERR

Fehlernummer des zuletzt aufgetretenen Fehlers im Programm ermitteln.

```
y=ERR
```

ERROR

Befehl zum Simulieren eines Laufzeitfehlers.

```
ERROR Fehlernummer
```

ON ERROR

Sprungmarke für Fehlerbehandlungsroutine festlegen.

```
ON ERROR GOSUB Sprungmarke
```

TROFF

Programmverfolgung mit Ausgabe der Zeilennummern ausschalten.

```
TROFF
```

TRON

Programmverfolgung mit Ausgabe der Zeilennummern einschalten.

```
TRON
```

4.2.10 Grafikanweisungen

CIRCLE

Kreis, Kreisausschnitt oder Ellipse zeichnen.

```
CIRCLE [STEP] (x,y), Radius[,Farbe[,Startwinkel,Endwinkel][,Aspekt]
```

DRAW

Makrosprache zum Zeichnen grafischer Figuren.

```
DRAW Zeichenkette
```

GET

Bildschirmausschnitt in ein Datenfeld kopieren.

```
GET (x1, y1)-(x2, y2), Datenfeld
```

LINE

Befehl zum Zeichnen von Linien und Rechtecken.

```
LINE [STEP] (x1,y1)-[STEP] (x2,y2)[,Farbe][,B[F]] [,Muster]
```

PAINT

Befehl zum Ausfüllen von geschlossenen Flächen.

```
PAINT (x,y) [[,Farbe][,Grenze][,Hintergrund]]
```

PALETTE/PALETTE USING

Befehle zur Definition und Zuordnung von Farbpaletten.

```
PALETTE Attribut, Farbe
PALETTE USING Ganzzahldatenfeld(Index)
```

PMAP

Funktion, um physikalische Koordinaten in globale umzurechnen und umgekehrt.

```
y=PMAP(x, Option)
```

POINT

Farbe eines Bildschirmpunkts bzw. Grafikcursor-Position ermitteln.

```
Farbe=POINT(x,y)
Position=POINT(Option)
```

PRESET

Einen Bildpunkt auf dem Grafikbildschirm löschen oder zeichnen.

```
PRESET [STEP] (x,y) [,Farbe]
```

PSET

Einen Bildpunkt auf dem Grafikbildschirm löschen.

```
PSET [STEP] (x,y) [,Farbe]
```

PUT

Inhalt eines Bildschirmausschnittes, der mit *GET* in ein Integer-Datenfeld gespeichert wurde, auf dem Grafikbildschirm ausgeben.

```
PUT [STEP] (x,y), Ganzzahldatenfeld [,Option]
```

VIEW

Zeichenfläche bzw. Grafikfenster auf dem Grafikbildschirm definieren.

```
VIEW [[SCREEN] [(x1,y1)-(x2,y2)[,[Farbe[,Grenze]]]
```

WINDOW

Globales Koordinatensystem für Grafikbildschirm definieren.

```
WINDOW [SCREEN] (x1,y1)-(x2,y2)
```

4.2.11 Konvertierfunktionen

ASC

ASCII-Code eines Zeichens ermitteln.

```
y=ASC(Zeichen)
```

ASCII

ASCII-Code des ersten Zeichens einer Zeichenkette ermitteln.

```
y=ASCII(Zeichenkette)
```

BIN$

Ganzzahlen (Bereich: –32768 bis 65535) in binäre Zeichenkette konvertieren.

```
a$=BIN$(Ausdruck)
```

CBCD

Numerischen Ausdruck in BCD-Fließkommazahl umwandeln.

```
y@@=CBCD(Ausdruck)
```

CDBL

Numerischen Ausdruck in doppeltgenaue Fließkommazahl umwandeln.

```
y#=CDBL(Ausdruck)
```

CEIL

Nächstgrößere oder gleichgroße Ganzzahl zu einem numerischen Ausdruck ermitteln.

```
y=CEIL(Ausdruck)
```

CEXT

Numerischen Ausdruck in Zahl mit erweiterter Genauigkeit umwandeln.

```
y##=CEXT(Ausdruck)
```

CFIX

Numerischen Ausdruck in BCD-Festkommazahl umwandeln.

```
y@=CFIX(Ausdruck)
```

CHR$

ASCII-Zeichen(kette) aus ASCII-Code(s) erzeugen.

```
a$=CHR$(Code,...)
```

CINT

Numerischen Ausdruck in Ganzzahl umwandeln.

```
y%=CINT(Ausdruck)
```

CLNG

Numerischen Ausdruck in lange Ganzzahl umwandeln.

```
y&=CLNG(Ausdruck)
```

CQUD

Numerischen Ausdruck in Quad-Ganzzahl umwandeln.

```
y&&=CQUD(Ausdruck)
```

CSNG

Numerischen Ausdruck in einfachgenaue Fließkommazahl umwandeln.

```
y!=CSNG(Ausdruck)
```

CVB

Zeichenkette aus einer Randomdatei in BCD-Fließkommazahl umwandeln.

```
y@@=CVB(10-Byte-Zeichenkette)
```

CVD

Zeichenkette aus einer Randomdatei in doppeltgenaue Fließkommazahl umwandeln.

```
y#=CVD(8-Byte-Zeichenkette)
```

CVE

Zeichenkette aus einer Randomdatei in Zahl mit erweiterter Genauigkeit umwandeln.

```
y##=CVE(10-Byte-Zeichenkette)
```

CVF

Zeichenkette aus einer Randomdatei in BCD-Festkommazahl umwandeln.

```
y@=CVF(8-Byte-Zeichenkette)
```

CVI

Zeichenkette aus einer Randomdatei in Ganzzahl umwandeln.

```
y%=CVI(2-Byte-Zeichenkette)
```

CVL

Zeichenkette aus einer Randomdatei in lange Ganzzahl umwandeln.

```
y&=CVL(4-Byte-Zeichenkette)
```

CVMD

Stringvariable im Microsoft-Format aus einer Randomdatei in doppeltgenaue Fließkommazahl umwandeln.

```
y#=CVMD(8-Byte-Microsoft-Zeichenkette)
```

CVMS

Stringvariable im Microsoft-Format aus einer Randomdatei in eine einfachgenaue Fließkommazahl umwandeln.

```
y!=CVMS(4-Byte-Microsoft-Zeichenkette)
```

CVQ

Zeichenkette aus einer Randomdatei in Quad-Ganzzahl umwandeln.

```
y&&=CVQ(8-Byte-Zeichenkette)
```

CVS

Zeichenkette aus einer Randomdatei in einfachgenaue Fließkommazahl umwandeln.

```
y!=CVS(4-Byte-Zeichenkette)
```

HEX$

Numerischen Ausdruck in hexadezimale Zeichenkette umwandeln.

```
a$=HEX$(Ausdruck)
```

INT

Ganzzahligen Anteil eines Argumentes ermitteln.

```
y=INT(Ausdruck)
```

MKB$

BCD-Fließkommazahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKB$(Ausdruck)
```

MKD$

Doppeltgenaue Fließkommazahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKD$(Ausdruck)
```

MKE$

Zahl erweiterter Genauigkeit in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKE$(Ausdruck)
```

MKF$

BCD-Festkommazahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKF$(Ausdruck)
```

MKI$

Ganzzahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKI$(Ausdruck)
```

MKL$

Lange Ganzzahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKL$(Ausdruck)
```

MKMD$

Doppeltgenaue Fließkommazahl in Microsoft-Zeichenkettenformat für Randomdatei umwandeln.

```
MSFeld$=MKMD$(Ausdruck)
```

MKMS$

Einfachgenaue Fließkommazahl in Microsoft-Zeichenkettenformat für Randomdateien umwandeln.

```
MSFeld$=MKMS$(Ausdruck)
```

MKQ$

Quad-Ganzzahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKQ$(Ausdruck)
```

MKS$

Einfachgenaue Fließkommazahl in Zeichenkette für Randomdatei umwandeln.

```
Feld$=MKS$(Ausdruck)
```

OCT$

Numerischen Ausdruck in oktale Zeichenkette umwandeln.

```
a$=OCT$(Ausdruck)
```

VAL

Zeichenkette in eine Fließkommazahl umwandeln.

```
y=VAL(Zeichenkettenausdruck)
```

4.2.12 Mathematische Funktionen

ABS

Absolutwert ermitteln.

ATN

Berechnung des Arcustangens (Bogenmaß)

```
y=ATN(Ausdruck)
```

COS

Berechnung des Cosinus (Bogenmaß).

```
y=COS(Ausdruck)
```

DECR

Numerische Variable um einen bestimmten Betrag abmindern.

```
DECR Variable [,Betrag]
```

EXP

Ermittlung der Potenz zur Basis e (2.7182...).

```
y=EXP(Ausdruck)
```

EXP2

Ermittlung der Potenz zur Basis 2.

```
y=EXP2(Ausdruck)
```

EXP10

Ermittlung der Potenz zur Basis 10.

```
y=EXP10(Ausdruck)
```

FIX

Nachkommastellen eines Argumentes abschneiden.

```
y=FIX(Ausdruck)
```

FIXDIGITS

Anzahl der Dezimalstellen von BCD-Festkommazahlen bestimmen oder lesen.

```
y=FIXDIGITS
FIXDIGITS=Ausdruck
```

INCR

Numerische Variable, um einen bestimmten Betrag erhöhen.

```
INCR Variable [,Betrag]
```

LOG

Berechnung des natürlichen Logarithmus (Basis e).

```
y=LOG(Ausdruck)
```

LOG2

Berechnung des Logarithmus zur Basis 2.

```
y=LOG2(Ausdruck)
```

LOG10

Berechnung des Logarithmus zur Basis 10.

```
y=LOG10(Ausdruck)
```

MAX%

Größten Wert von Integer-Werten ermitteln.

```
y%=MAX%(Argument,...)
```

MAX

Größten Wert beliebiger numerischer Argumente ermitteln.

```
y=MAX(Argument,...)
```

MIN%

Kleinsten Wert von Integer-Werten ermitteln.

```
y%=MIN%(Argument,...)
```

MIN

Kleinsten Wert beliebiger numerischer Argumente ermitteln.

```
y=MIN(Argument,...)
```

RANDOMIZE

Startwert für Zufallszahlengenerator setzen.

```
RANDOMIZE (Ausdruck)
```

RND

Zufallszahl ermitteln.

```
y=RND(Ausdruck)
```

ROUND

Funktion zur Rundung numerischer Werte auf eine bestimmte Anzahl von Dezimalzahlen.

```
y=ROUND(Ausdruck, Stellenanzahl)
```

SGN

Vorzeichen ermitteln.

```
y=SGN(Ausdruck)
```

SIN

Berechnung des Sinus (Bogenmaß).

```
y=SIN(Ausdruck)
```

SQR

Wurzelfunktion.

```
y=SQR(Ausdruck)
```

TAN

Berechnung des Tangens (Bogenmaß).

```
y=TAN(Ausdruck)
```

4.2.13 Musik

BEEP

Ton erzeugen (insbesondere als Warnsignal in Anwendungsprogrammen).

```
BEEP [Anzahl]
```

SOUND

Beliebige Töne erzeugen.

```
SOUND Frequenz, Dauer
```

4.2.14 Programmablaufsteuerung

CALL

Benutzerdefiniertes Unterprogramm aufrufen.

```
CALL Unterprogrammname[(Parameter,...)]
```

CALL ABSOLUTE

Maschinenprogramm aufrufen (wird z.B. zur Konvertierung von GW-Basic-Programmen benötigt.

```
CALL ABSOLUTE Adresse (Parameter)
```

CALL INTERRUPT

Aufruf eines System-Interrupts.

```
CALL INTERRUPT Nummer
```

CHAIN

Chain-Modul (EXE- oder PBC-Datei) nachladen und ausführen (nicht innerhalb der Entwicklungsumgebung ausführbar!).

```
CHAIN Dateiname
```

COMMON

Variablen für Chain-Module als global deklarieren.

```
COMMON Variable,...
```

DECLARE

Benutzerdefinierte Funktionen und Unterprogramme aus Objektdateien oder Units deklarieren.

```
DECLARE SUB|FUNCTION Name [(Parameter,...)]
```

DEF FN/END DEF

Ein- oder mehrzeilige Funktionen definieren (diese Funktionen sind nur lokal innerhalb eines Moduls nutzbar!).

```
DEF FNname [(Parameter,...)]=Ausdruck
DEF FNname [(Parameter,...)]
 : 'Anweisungen
END DEF
```

DO-LOOP

Schleifenkonstrukt.

```
DO [Bedingung]
 :'Anweisungen
LOOP [Bedingung]
```

END

Programm beenden und zum Betriebssystem zurückkehren. Über den optionalen Parameter kann ein Wert an das Betriebssystem zurückgegeben werden (in der Regel DOS-Fehlercode).

```
END [Rückgabewert]
```

EXECUTE

Programm beenden und angegebenes Programm laden und ausführen.

```
EXECUTE Dateiname
```

EXIT

Blockstruktur vorzeitig beenden.

```
EXIT FUNCTION|SELECT|DEF|FOR|IF|LOOP|SUB
```

FOR-NEXT

Programmschleife. Wird keine Schrittweite angegeben, wird standardmäßig 1 genommen.

```
FOR Zählvariable= Anfangswert TO Endwert [STEP Schrittweite]
  :'Anweisungen
NEXT Zählvariable
```

FUNCTION/END-FUNCTION

Definition einer benutzerdefinierten Funktion.

```
FUNCTION Name [(Parameter,...)] [PUBLIC|PRIVATE]
  :'Anweisungen
  Name=Ergebnis
END FUNCTION
```

GOSUB

Unterprogramm über Sprungmarke aufrufen.

```
GOSUB Sprungmarke
```

GOTO

Programm an angegebener Sprungmarke fortsetzen.

```
GOTO Sprungmarke
```

IF

Entscheidungsabfrage.

```
IF Bedingung THEN Anweisung[en] [ELSE Anweisung[en]]
```

IF-ELSE-ENDIF

Blockstruktur der Entscheidungsabfrage.

```
IF 1.Bedingung THEN
    :'Anweisungen
  [ELSEIF 2.Bedingung THEN
    :'Anweisungen]
  [ELSEIF n.Bedingung THEN
    :'Anweisungen]
  [ELSE
    :'Anweisungen]
END IF
```

LET

Diese Zuweisung ist nur aus Kompatibilitätsgründen zu älteren Basic-Dialekten implementiert und sollte in neueren Programmen nicht mehr eingesetzt werden (Variablenzuweisung).

```
LET Variable=Ausdruck
```

ON-GOSUB

Ein Unterprogramm abhängig von einem Wert über Sprungmarke aufrufen.

```
ON Wert GOSUB Sprungmarke[,Sprungmarke,...]
```

ON/GOTO

Die Programmausführung abhängig von einem Wert an der angegebenen Sprungmarke fortsetzen.

```
ON Wert GOTO Sprungmarke[,Sprungmarke,...]
```

REG

Befehl zum Setzen der Registerwerte für einen Interrupt-Aufruf.

```
REG Register, Wert
```

REG

Funktion zum Auslesen von Registerwerten nach einem Interrupt-Aufruf.

```
y=REG(Register)
```

REM

Einleitung von Kommentarzeilen (alternativ durch »'« möglich).

```
REM Kommentar
```

RESTORE

DATA-Zeiger für den nächsten *READ*-Befehl auf ein Anfangselement einer angegebenen Sprungmarke setzen.

```
RESTORE Sprungmarke
```

RESUME

Programmausführung nach einer Fehlerbehandlungsroutine fortsetzen.

```
RESUME 0|NEXT|Sprungmarke
```

RETURN

Unterprogramm, welches mit *GOSUB* aufgerufen wurde, beenden und an Stelle nach dem Aufruf zurückkehren bzw. Programm an angegebener Sprungmarke fortsetzen ...

```
RETURN [Sprungmarke]
```

RUN

Aktuelles Programm neu starten bzw. neues Programm ausführen (keine Variablenübergabe wie bei *CHAIN* möglich).

```
RUN [Dateiname]
```

SEEK

Position des *DATA*-Zeigers in einer Random- oder Binärdatei ermitteln bzw. setzen.

```
y=SEEK(Dateinummer)
SEEK #Dateinummer, Position
```

SELECT-CASE

Blockstruktur (Auswahl).

```
SELECT CASE Ausdruck
  CASE 1.Bedingung
    :'Anweisungen
 [CASE 2.Bedingung
    :'Anweisungen]
 [CASE n.Bedingung
    :'Anweisungen]
 [CASE ELSE
    :'Anweisungen]
END SELECT
```

SHELL

Ausführen eines DOS-Befehls bzw. Starten eines Maschinenprogramms, wobei das Programm im Speicher verbleibt. Nach der Ausführung dieser Anweisung wird das Programm anders als bei *EXECUTE* fortgesetzt.

```
SHELL [Befehl]
```

STOP

Programm abbrechen.

```
STOP
```

SUB/END-SUB

Definition eines benutzerdefinierten Unterprogramms, das später über *CALL* aufrufbar ist.

```
SUB Name [(Parameter,...)] [PUBLIC|PRIVATE]
  :'Anweisungen
END SUB
```

SUB-INLINE/END-SUB

Definition eines Unterprogramms, das nur aus Assembler-Code besteht.

```
SUB Name INLINE
  $INLINE ...
END SUB
```

SYSTEM

Programm beenden. Dieser Befehl ist nur aus Kompatibilitätsgründen zu älteren Basic-Dialekten in Power Basic vorhanden.

```
SYSTEM
```

WAIT

Auf Wert vom I/O-Port warten.

```
WAIT Port, n, m
```

WHILE-WEND

Schleifenkonstruktion. Die Schleife wird so lange ausgeführt, bis die bestimmte Bedingung erfüllt ist.

```
WHILE Bedingung
  :'Anweisungen
WEND
```

4.2.15 Speicherverwaltung und Speicherfunktionen

BLOAD

Speicherblock aus einer Datei lesen.

```
BLOAD Dateiname [,Adresse]
```

BSAVE

Speicherblock in eine Datei speichern.

```
BSAVE Dateiname, Adresse, Länge
```

CLEAR

Alle Variablen zurücksetzen und Ereignisverfolgung abschalten.

```
CLEAR
```

DEF-SEG

Datensegment für Speicherfunktionen festlegen.

```
DEF SEG [= Segment]
```

ENDMEM

Höchste unter DOS verfügbare Speicheradresse ermitteln.

```
y=ENDMEM
```

FRE

Freien Speicherplatz ermitteln.

```
y&=FRE(""|0|-1|-2)
```

MEMSET

Obere Speichergrenze für Power Basic festlegen.

```
MEMSET Adresse
```

PEEK

Einzelnes Byte aus dem Speicher lesen.

```
y%=PEEK(Adresse)
```

PEEKI

Speicherwort lesen (2 Byte).

```
y%PEEK(Adresse)
```

PEEKL

4 Byte aus dem Speicher lesen.

```
y&=PEEKL(Adresse)
```

PEEK$

Bytefolge aus dem Speicher lesen.

```
y$=PEEK$(Adresse, Anzahl)
```

POKE

Einzelnes Byte in den Speicher schreiben.

```
POKE Adresse, Byte
```

POKEI

Speicherwort (2 Byte) in den Speicher schreiben.

```
POKEI Adresse, Ganzzahl
```

POKEL

Vier Byte in den Speicher schreiben.

```
POKEL Adresse, LangeGanzzahl
```

POKE$

Bytefolge in den Speicher schreiben.

```
POKE$ Adresse, Zeichenkette
```

STRPTR

Offset einer Zeichenkettenvariablen ermitteln.

```
y=STRPTR(Zeichenkette)
```

STRSEG

Speichersegment einer Zeichenkette ermitteln.

```
y=STRSEG(Zeichenkette)
```

VARPTR

Offset einer Variablen ermitteln.

```
y=VARPTR(Variable)
```

VARPTR$

Adresse einer Variablen in Zeichenkettenform ermitteln.

```
a$=VARPTR$(Variable)
```

VARSEG

Speichersegment einer Variablen ermitteln.

```
y=VARSEG(Variable)
```

4.2.16 Stringoperationen

EXTRACT$

Teilzeichenkette ab einem angegebenen Zeichen oder einer angegebenen Zeichenkette ermitteln.

```
a$=EXTRAC$(Zeichenkette, [ANY] Teilzeichenkette)
```

FLEXCHR$

Zeichen zum Auffüllen des Dateipuffers für Direktzugriffsdateien festlegen.

```
FLEXCHR$=Zeichen
y$=FLEXCHR$
```

INSTR

Position einer Teilzeichenkette in einer Zeichenkette ermitteln.

```
y=INSTR([n,] Zeichenkette, [ANY] Teilzeichenkette)
```

LCASE$

Zeichenketten in Kleinbuchstaben umwandeln.

```
a$=LCASE$(Zeichenkette)
```

LEFT$

Linken Teil einer Zeichenkette ermitteln.

```
a$=LEFT$(Zeichenkette, Länge)
```

LEN

Länge einer Zeichenkette ermitteln.

```
y=LEN(Zeichenkette)
```

LTRIM$

Führende Zeichen bzw. Leerzeichen von einer Zeichenkette abspalten.

```
a$=LTRIM$(Zeichenkette [,[ANY] Teilzeichenkette])
```

MAX$

Zeichenkette mit den größten ASCII-Codes ermitteln.

```
y$=MAX$(Zeichenkette,...)
```

MID$

Funktion, um eine Teilzeichenkette aus einer Zeichenkette zurückzuliefern.

```
a$=MID$(Zeichenkette, Startposition [,Länge])
```

MID$

Befehl, um eine Teilzeichenkette in einer Zeichenkette zu ersetzen.

```
MID$(Zeichenkette, Startposition [,Länge])=NeueTeilzeichenkette
```

MIN$

Zeichenkette mit den kleinsten ASCII-Codes ermitteln.

```
y$=MIN$(Zeichenkette,...)
```

REMOVE$

Zeichen bzw. Teilzeichenkette aus einer Zeichenkette entfernen.

```
a$=REMOVE$(Zeichenkette, [ANY] Teilzeichenkette)
```

REPEAT$

Zeichenkette aufbauen, die eine Vorgabezeichenkette n-mal enthält.

```
a$=REPEAT$(n, Zeichenkette)
```

REPLACE

Ersetzt eine Teilzeichenkette in einer Zeichenkette durch eine andere.

```
REPLACE [ANY] Teilzeichenkette WITH NeueZeichenkette IN Zeichenkette
```

RIGHT$

Rechten Teil einer Zeichenkette zurückliefern.

```
a$=RIGHT$(Zeichenkette, Länge)
```

RTRIM$

Rechte Zeichen bzw. Leerzeichen von einer Zeichenkette abspalten.

```
a$=RTRIM$(Zeichenkette [,[ANY] Teilzeichenkette])
```

SPACE$

Zeichenkette mit n Leerzeichen zurückliefern.

```
a$=SPACE$(n)
```

STR$

Zahl in eine Zeichenkette umwandeln.

```
a$=STR$(Zahl)
```

STRING$

Zeichenkette aufbauen, die eine bestimmte Anzahl gleicher Zeichen enthält.

```
a$=STRING$(Anzahl, ASCII-Code|Zeichen)
```

TALLY

Anzahl von Zeichen bzw. Teilzeichenketten in einer Zeichenkette ermitteln.

```
y=TALLY(Zeichenkette, [ANY] Teilzeichenkette)
```

UCASE$

Zeichenkette in Großbuchstaben umwandeln.

```
a$=UCASE$(Zeichenkette)
```

USING$

Formatstring definieren.

```
a$=USING$(FormatZeichenkette, Ausdruck)
```

VERIFY

Zeichenkettenvergleich. Überprüfung, ob sämtliche Zeichen eines Strings in einem anderen enthalten sind.

```
y=VERIFY([Start,]Zeichenkette, Teilzeichenkette)
```

4.2.17 Textbildschirm

CLS

Bildschirm bzw. mit VIEW definierte Zeichenfläche löschen.

```
CLS
```

COLOR

Bildschirmfarben im Grafikmodus festlegen.

```
COLOR Hintergrund, Palette
COLOR Vordergrund, Hintergrund
```

COLOR

Bildschirmfarben im Textmodus festlegen.

```
COLOR [Vordergrund][,[Hintergrund][,Rahmen]]
```

CSRLIN

Aktuelle Zeile des Textcursors ermitteln.

```
y=CSRLIN
```

LOCATE

Textcursor auf dem Bildschirm positionieren.

```
LOCATE [Zeile],[Spalte][,Cursor][,Start][,Stop]
```

SCREEN

Funktion zum Ermitteln des ASCII-Codes bzw. des Attributes eines bestimmten Zeichens auf dem Bildschirm.

```
y=SCREEN(Zeile, Spalte [,Option])
```

SCREEN

Befehl zum Setzen des Bildschirmmodus.

```
SCREEN [Modus] [[,Farbe][,Zielseite][,AktiveSeite]]
```

4.3 Schnellinformation: Wie kann ich ...?

In diesem Kapitel werden einige Lösungswege für Programmierprobleme besprochen, die in der Praxis regelmäßig auftauchen. Hier haben Sie die Möglichkeit, sich Anregungen und Hinweise für Ihre eigenen Anwendungen zu holen.

Dateiverwaltung

... beliebige Suchpfade für Arbeitsdateien in einem Programm berücksichtigen?

Führen Sie in Ihrem Programm eine Variable ein, in der das aktuelle Arbeitsverzeichnis gespeichert wird. Verwenden Sie anschließend für alle *OPEN*-Befehle eine Kombination aus Suchpfad und Dateinamen.

```
'Arbeitspfad setzen
Verzeichnis$="C:\HAUSHALT\"
'Datei im Suchpfad setzen
OPEN Verzeichnis$+"TEST.DAT" FOR INPUT AS #1
```

... herausfinden, ob eine Datei überhaupt existiert?

Über die integrierte Power-Basic-Funktion *DIR$* können Sie nicht nur Dateien des Verzeichnisses ermitteln, sondern auch feststellen, ob eine Datei überhaupt existiert.

```
Dateiname$="TEST.DAT"
IF LEN(DIR$(Dateiname$)>0 THEN
    'Datei ist vorhanden, wenn die Länge des
    'übereinstimmenden Dateinamens ungleich Null ist
  ELSE
    'Datei ist nicht vorhanden
END IF
```

... überprüfen, ob die richtige Diskette im Diskettenlaufwerk liegt?

Geben Sie der Diskette über die DOS-Anweisung *LABEL* eine Bezeichnung; diese können Sie über die *DIR$*-Funktion abfragen. Erst wenn der korrekte Name existiert, können Sie die Arbeiten mit dem Laufwerk zulassen. Diese Möglichkeit ließe sich speziell für Installationsprogramme nutzen.

```
'Laufwerksbezeichnung A
Bezeichnung$=("A:\", 8)
'Laufwerksbezeichnung B
Bezeichnung$=("B:\", 8)
'aktuelles Laufwerk
Bezeichnung$=("\", 8)
```

... eine sequentielle Datei komplett in den Speicher einlesen?

Über eine *WHILE-WEND*-Schleife, die *EOF*-Funktion und ein Datenfeld kann der Inhalt einer sequentiellen Datei in den Speicher eingelesen werden. Bedenken Sie, daß der Arbeitsspeicher begrenzt ist, so daß die Dateien nur bis zu einer begrenzten Größe verarbeitet werden können. In unserem folgenden Beispiel gehen wir davon aus, daß maximal 200 Textzeilen in der einzulesenden Datei vorhanden sind.

```
DIM Inhalt$(200)

DNr=FREEFILE
Dateiname$="TEST.TXT"
OPEN Dateiname$ FOR INPUT AS #DNr
Zeile%=1
```

```
WHILE NOT EOF(DNr)
  LINE INPUT #DNr, Inhalt$(Zeile%)
  DECR Zeile%
WEND
INCR Zeile%
```

DOS

... Systemvariablen in eigenen Programmen einsetzen?

Systemvariablen des Betriebssystems werden auf DOS-Ebene durch den Befehl *SET* gesetzt und können durch die *ENVIRON$*-Funktion abgefragt werden. Nehmen wir an, die Systemvariable *EMS* enthält Informationen, wieviel EMS-Speicher in Kbyte maximal verwendet werden darf. Auf DOS-Ebene könnte dann der Befehl *SET EMS=64* auf Kommandozeilenebene bzw. über die AUTOEXEC.BAT (vgl. Handbuch zum Betriebssystem) eingegeben werden. Die nachfolgenden Anweisungen erlauben dann einem Anwendungsprogramm diese unter DOS definierte Variable im Zeichenkettenformat einzulesen.

```
'Systemvariable auslesen
Variable$=ENVIRON$("EMS")
'in numerisches Äquivalent
Variable%=VAL(Variable$)
'nun Weiterverwertung möglich
```

Auf diese Art und Weise können Sie sowohl den Arbeitspfad für Ihr Programm, Ihre Grafikkarte oder auch die Existenz eines Coprozessors anmelden.

... eine Arbeitsdatei direkt beim Programmstart an ein Anwendungsprogramm übergeben?

Will man eine Arbeitsdatei direkt beim Programmstart mit angeben, kann dies ohne spezielle Programmierarbeit geleistet werden. Der Aufruf könnte für unser Programm HAUSHALT und die Datei HAUSHALT.DAT wie folgt aussehen.

```
HAUSHALT HAUSHALT.DAT [Return]
```

Da in den meisten Fällen ein Standardsuffix angehängt wird, ist es sinnvoller, die Datei nur ohne Suffix anzugeben.

```
HAUSHALT HAUSHALT [Return]
```

In den nachfolgenden Zeilen wird die Zeichenkette, die beim Programmstart angegeben wird, über die *COMMAND$*-Funktion ausgewertet.

```
'Kommandostring einer Variablen zuweisen
Datei$=COMMAND$
IF LEN(Datei$)=0 THEN
   'keine Datei gewählt
   Datei$="keine"
  ELSEIF RIGHT$(Datei$)<>".DAT"
   'u.U. Dateikürzel anhängen
   Datei$=Datei$+".DAT"
END IF
```

... Dateien vor Schreibzugriffen sichern?

Verwenden Sie den *ATTRIB*-Befehl, um eine Datei gegen Zugriffe zu schützen.

```
ATTRIB Dateiname$, 1
```

... Dateien innerhalb eines Dateiverzeichnisses verstecken?

Auch das Verstecken von Dateien ist über den ATTRIB-Befehl von Power Basic möglich. Benutzen Sie dazu die folgende Anweisung.

```
ATTRIB Dateiname$, 2
```

Druckerausgabe

... den Drucker so programmieren, daß bei einer kleinen Schrift auch mehr als 80 Zeichen in einer Zeile ausgebbar sind?

Standardmäßig beträgt die Zeilenbreite des Druckers 80 Zeichen. Um mehr Zeichen in einer Zeile auszugeben, ist der *WIDTH*-Befehl zu verwenden.

```
'Zeilenbreite auf 150 Zeichen festlegen
WIDTH "LPT1", 150
'Zeilenbreite nicht länger kontrollieren
WIDTH "LPT1", 255
```

Bei diesem Beispiel ist der Drucker an der ersten parallelen Schnittstelle angeschlossen.

... Druckersteuerzeichen zum Drucker senden, ohne jedesmal einen Zeilenvorschub auszulösen?

Um Druckersteuerzeichen ohne Formfeed (Seitenvorschub) zum Drucker zu senden, ist *LPRINT* mit einem abschließenden Semikolon zu benutzen. Eine Anweisung für den Star SG-10/15 zum Einschalten des Doppeldrukkes hat dann nachfolgende Form.

```
LPRINT CHR$(27)"G";
```

... einen Seitenvorschub bzw. einen Zeilenvorschub bei einem Drucker (Matrix- und Laserdrucker) auslösen?

Sowohl den Seitenvorschub als auch den Zeilenvorschub kann man über *LPRINT* auslösen. Die Steuerbefehle hierzu sind beim Matrix- und Laserdrucker identisch.

```
'Zeilenvorschub (kein Semikolon!)
LPRINT
'Seitenvorschub
LPRINT CHR$(12)
```

Fehlerbehandlung

... tun, wenn auf eine Datei nicht mehr zugegriffen werden kann?

Wahrscheinlich haben Sie versucht, die Datei ein zweites Mal zu öffnen oder eine Dateinummer beim Öffnen ein zweites Mal eingesetzt. Über die *RESET*-Anweisung können Sie in einem solchen Fall sämtliche Dateien schließen, ohne das Programm beenden zu müssen.

... in einem Programm eigene Fehler definieren, die über die Fehlerbehandlung verfolgt werden?

Um einen Fehler zu simulieren, können Sie die *ERROR*-Anweisung verwenden. Der Fehlercode darf einen Wert zwischen 0 und 255 einnehmen. Ein Beispielprogramm zum Testen ist nachfolgend abgedruckt.

```
ON ERROR GOTO Fehler
CLS
'nicht belegte Fehlernummern:
'1/8/12/16/17/18/21/22/23/26/
'28-49/56/59/60/65/66/77-200/
'202/207-241/245-255
'Fehler Nr.1 simulieren
```

```
ERROR 1
'Fehler Nr.2 simulieren
ERROR 2
END

Fehler:
  SELECT CASE ERR
    CASE 1
      'Fehlernummer noch nicht von
      'Power Basic belegt
      PRINT ERR;"> Feierabend!"
    CASE 2
      'neu belegt: normal "Syntax Error"
      PRINT ERR;"> Kaffeepause!"
    CASE ELSE
      PRINT ERR;"> Fehler unbekannt"
  END SELECT
RESUME NEXTî
```

Grafik

... mathematische Funktionen zeichnen?

Berechnen Sie für jede x/y-Koordinate den Bildpunkt und setzen ihn einzeln über die Funktion *PSET*.

... unterschiedliche Linienarten definieren?

Der LINE-Befehl zum Zeichnen von Linien erlaubt die direkte Definition von Linienarten. Die einfachste Möglichkeit ist, die Linie mit einem binären Muster zu definieren. Jedes Zeichen entspricht dabei genau einem Pixel. Eine Angabe in einem anderen Zahlensystem, z.B. hexadezimal, ist ebenso möglich. In den nachfolgenden Beispielen sind einfache gepunktete Linien und strichpunktierte Linien definiert. Versuchen Sie einmal eine Änderung der binären Konstanten (16stellig).

```
'unbedingt in Grafikmodus schalten!
SCREEN 9
'einfache gepunktete Linien
LINE (20,10)-(600,10),,,&B1000000000000000
LINE (20,20)-(600,20),,,&B1000000010000000
LINE (20,30)-(600,30),,,&B1000100010001000
LINE (20,40)-(600,40),,,&B1010101010101010
'Linien und Punkte
LINE (20,50)-(600,50),,,&B1001110011100111
LINE (20,60)-(600,60),,,&B1110101111010111
LINE (20,70)-(600,70),,,&B1111101011111010
LINE (20,80)-(600,80),,,&B1111111111111010
```

Konvertierroutinen

... fehlende Konvertierroutinen in Power Basic implementieren?

Power Basic stellt über die Funktionen *BIN$, HEX$* und *OCT$* bereits Möglichkeiten zur Verfügung, numerische Werte in Zeichenketten eines anderen Zahlensystems zu konvertieren. Um diese wieder in Zahlenwerte zurückzuübersetzen, müssen spezielle benutzerdefinierte Funktionen erstellt werden. Da Power Basic auch Konstante dieser Zahlensysteme zuläßt, ist eine Umsetzung über die *VAL*-Funktion kein Problem.

```
'Binärstring in numerischen Wert
FUNCTION BIN2DEZ&(Binaer$)
  BIN2DEZ&=VAL("&B"+Binaer$)
END FUNCTION

'Oktalstring in numerischen Wert
FUNCTION OCT2DEZ&(Oktal$)
  OCT2DEZ&=VAL("&O"+Oktal$)
END FUNCTION

'Hexadezimalstring in numerischen Wert
FUNCTION HEX2DEZ&(Hexadezimal$)
  HEX2DEZ&=VAL("&H"+Hexadezimal$)
END FUNCTION
```

Den Einsatz dieser Funktionen können Sie anhand des folgenden Beispielprogramms nachvollziehen.

```
'Beispielprogramm
CLS
a%=226
a$=BIN$(a%)
b$=OCT$(a%)
c$=HEX$(a%)
PRINT BIN$(a%)
PRINT BIN2DEZ&(a$)
PRINT OCT$(a%)
PRINT OCT2DEZ&(b$)
PRINT HEX$(a%)
PRINT HEX2DEZ&(c$)
```

Funktionen

... eigene mathematische Funktionen definieren, die sich wie interne Power-Basic-Funktionen verhalten?

Zunächst ist anhand einer Formelsammlung zu ermitteln, wie sich die Funktion mit Hilfe der internen Power-Basic-Anweisungen ausdrücken läßt. Anschließend erzeugen Sie eine echte Funktion mit Hilfe der Schlüsselwörter *FUNCTION/END-FUNCTION*. Dabei ist zu berücksichtigen, welchen Datentyp und damit mit welcher Genauigkeit das Ergebnis ermittelt werden soll. Im nachfolgenden Beispiel errechnen wir den Cotangens mit einfacher Genauigkeit. Der Aufruf dieser Funktion erfolgt wie der Aufruf der internen *SIN*- oder *COS*-Funktion.

```
FUNCTION COT!(X!)
  COT = 1/TAN(X!)
END FUNCTION
```

Textformatierung auf dem Bildschirm

... Text links-, rechtsbündig oder zentriert auf dem Bildschirm ausgeben?

Über die integrierten Stringfunktionen läßt sich Text sehr einfach formatiert auf dem Bildschirm ausgeben. Dazu ist lediglich die Kenntnis über die Spaltenanzahl des Bildschirms erforderlich, die im normalen Textmodus 80 Zeichen beträgt. Da Texte über den *PRINT*-Befehl immer von links nach rechts ausgegeben werden, muß die Zeichenkette mit Leerzeichen zunächst aufbereitet werden.

```
Text$="Dies ist die Ausgabe!"
Laenge%=LEN(Text$)
'linksbündige Ausgabe
Zeile%=1
LOCATE Zeile%, 1
PRINT Text$;
'rechtsbündige Ausgabe
LOCATE Zeile%+1, 1
PRINT SPACE$(80-Laenge%)+Text$
'zentrierte Ausgabe
LOCATE Zeile%+2, 1
PRINT SPACE$((80-Laenge%)\2)+Text$1
```

... ein Bildschirmfenster in der Bildschirmmitte ausgeben?

Mit Hilfe der Integer-Division und der Spalten- und Zeilenzahl des aktiven Bildschirms läßt sich das Bildschirmzentrum sehr einfach errechnen.

```
'mittlere Bildschirmzeile
MZeile%=25\2
'mittlere Spalte
MSpalte%=80\2
```

Von dieser Zentrumsposition aus kann nun das Bildschirmfenster, abhängig von Fensterbreite und -tiefe, plaziert werden. In der Regel errechnet man die Position der linken oberen Bildschirmecke.

```
Breite%=20
Tiefe% =10
'ZeilenPosition
ZPosition%=MSpalte% - (Tiefe%\2)
'Spaltenposition
SPosition%=MSpalte% - (Breite%\2)
```

Im Anschluß daran kann mit *LOCATE* und *PRINT* das Bildschirmfenster ausgegeben werden.

... eine Meldung an einer beliebigen Cursorposition ausgeben, ohne daß die ursprüngliche Cursorposition verlorengeht?

Merken Sie sich, bevor Sie die Meldung ausgeben, zunächst die alte Cursorposition mit den Funktionen *CSRLIN* und *POS*. Mit den ermittelten Werten können Sie nach der Textausgabe den Cursor wieder an die ursprüngliche Bildschirmkoordinate plazieren.

```
'alte Zeilenposition ermitteln
AltZeile%=CSRLIN
'alte Spaltenposition ermitteln
AltSpalte%=POS
'Text ausgeben in Zeile 1 und Spalte 1
LOCATE 1, 1
PRINT "Dies ist die Meldung!"
'an ursprüngliche Position zurückkehren
LOCATE AltZeile%, AltSpalte%
```

Anhang

A Quellen zum Programm HAUSHALT

Auf den folgenden Seiten sind noch einmal alle Quelltexte zusammengestellt, die Sie für das Programm HAUSHALT benötigen. Da sind zunächst die Headerdatei TOOLS.H und die zugehörige Funktionsbibliothek TOOLS.BAS, die wir in unserer ersten Sitzung erstellt haben, und anschließend die Datei HAUSHALT.BAS, die wir im Verlaufe des Tutoriums entwickelt haben. Beachten Sie den in einigen Zeilen auftretenden Tiefstrich, der in Power Basic als Fortsetzungszeichen für eine Zeile definiert ist.

Anhand der Listings können Sie noch einmal den Aufbau der einzelnen Dateien, vor allen Dingen aber der Datei HAUSHALT.BAS nachvollziehen. Beachten Sie die eingehaltene Ordnung. Nach dem Hauptprogrammteil und den Sprungmarken mit speziellen Programmanweisungen folgt der Unterprogramm- und Funktionsteil.

```
'Header-Datei: TOOLS.BAS
'Deklarationen der Routinen aus der Unit TOOLS

DECLARE SUB InfoZeile (STRING, INTEGER, INTEGER, INTEGER)
DECLARE FUNCTION SollLen$ (STRING, INTEGER)
DECLARE SUB Hintergrund (INTEGER, INTEGER)
DECLARE SUB Eingabe (STRING, STRING, INTEGER, INTEGER, INTEGER, INTEGER,_
        INTEGER)
DECLARE SUB Fenster (INTEGER, INTEGER, INTEGER, INTEGER, INTEGER, INTEGER)
DECLARE SUB Auswahl (STRING ARRAY, INTEGER, INTEGER, INTEGER, INTEGER)
```

Listing 1: TOOLS.H

```
'*******************************************************************
'* UNIT: TOOLS.BAS                                                 *
'* allgemein einsetzbare Funktionen und Unterprogramme             *
'*******************************************************************

SUB Auswahl (Text$(), Anzahl%, vfarbe%, hfarbe%, Wahl%) PUBLIC
  COLOR vfarbe%, hfarbe%        'Fensteraufbau
  Breite% = 10
  FOR x% = 1 TO Anzahl%         'max. Stringlänge
    IF LEN(Text$(x%)) > Breite% THEN Breite% = LEN(Text$(x%))
  NEXT x%
  FOR y% = 1 TO Anzahl%         'einheitl. Stringlänge
    Text$(y%) = SollLen$(Text$(y%), Breite%)
  NEXT y%
  Breite% = Breite% + 2         'Fenster
  CALL Fenster(5, 5, Breite%, Anzahl%, vfarbe%, hfarbe%)
  FOR x% = 1 TO Anzahl%
    IF x% = 1 THEN
      COLOR hfarbe%, vfarbe%
    ELSE
      COLOR vfarbe%, hfarbe%
    END IF
    LOCATE 5 + x%, 7, 0
    PRINT Text$(x%);
  NEXT x%                       'Auswahl
  BalkenPosition% = 1           'erster Eintrag aktiv
  WHILE a$ <> CHR$(13)
    a$ = INKEY$
    IF LEN(a$) = 2 THEN
      AltPosition% = BalkenPosition%
      Taste% = ASC(RIGHT$(a$, 1))
      SELECT CASE Taste%
        CASE 72                 'Cursor Up
          IF BalkenPosition% = 1 THEN
            BalkenPosition% = Anzahl%
          ELSE
            BalkenPosition% = BalkenPosition% - 1
          END IF
        CASE 80                 'Cursor Down
          IF BalkenPosition% = Anzahl% THEN
            BalkenPosition% = 1
          ELSE
            BalkenPosition% = BalkenPosition% + 1
          END IF
        CASE ELSE
      END SELECT                'Balken aktualisieren
      IF Taste% = 80 OR Taste% = 72 THEN
        COLOR vfarbe%, hfarbe% 'alter Balken
        LOCATE 5 + AltPosition%, 7, 0
        PRINT Text$(AltPosition%);
        COLOR hfarbe%, vfarbe% 'neuer Balken
        LOCATE 5 + BalkenPosition%, 7, 0
        PRINT Text$(BalkenPosition%);
      END IF
    END IF
```

```
    IF a$ <> CHR$(13) THEN a$ = ""
  WEND
  LOCATE , , 1
  Wahl% = BalkenPosition%
END SUB

SUB Eingabe (Vorgabe$, Eingab$, Laenge%, Zeile%, Spalte%, vfarbe%,_
             hfarbe%) PUBLIC
  IF LEFT$(Vorgabe$, 2) = "I|" THEN        'Beschränkung auf Ganzzahlen
      Zulaessig$ = "-+1234567890"
      Vorgabe$ = RIGHT$(Vorgabe$, LEN(Vorgabe$) - 2)
    ELSEIF LEFT$(Vorgabe$, 2) = "R|" THEN  'Fließkommazahl
      Zulaessig$ = "-+.,1234567890"
      Vorgabe$ = RIGHT$(Vorgabe$, LEN(Vorgabe$) - 2)
  END IF
  LOCATE Zeile%, Spalte%
  COLOR vfarbe%, hfarbe%
  PRINT Vorgabe$;
  COLOR hfarbe%, vfarbe%
  PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
  MinSpalte% = LEN(Vorgabe$) + Spalte%
  MaxSpalte% = LEN(Vorgabe$) + Spalte% + Laenge% - 1
  EditSpalte% = LEN(Vorgabe$) + Spalte% + LEN(Eingab$)
  IF EditSpalte%>MaxSpalte% THEN EditSpalte%= EditSpalte% - 1
  WHILE Zeichen$ <> CHR$(13)
    LOCATE Zeile%, EditSpalte%, 1
    Zeichen$ = INKEY$
    IF LEN(Zeichen$) = 1 THEN
      SELECT CASE ASC(Zeichen$)
        CASE 8   'Backspace
          IF EditSpalte% > MinSpalte% THEN
            IF LEN(Eingab$) = Laenge% THEN
              EditSpalte% = EditSpalte% + 1
            END IF
            Eingab$ = LEFT$(Eingab$, LEN(Eingab$) - 1)
            LOCATE Zeile%, MinSpalte%
            PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
            IF MinSpalte% < EditSpalte% THEN
              EditSpalte% = EditSpalte% - 1
            END IF
          END IF
        CASE 32 TO 254
          IF Zeichen$ = "." OR Zeichen$ = "," THEN
            IF INSTR(Eingab$, ANY ".,") > 0 THEN   'maximal ein Komma
                Zeichen$ = ""
              ELSE
                Zeichen$ = "."                     'Komma intern
            END IF
          END IF
          IF Zulaessig$ <> "" THEN                 'Zeichen zulässig?
            IF INSTR(Zulaessig$, Zeichen$) = 0 THEN
                Zeichen$ = ""                      'nein!
              ELSEIF EditSpalte% > MinSpalte% THEN 'nicht in der Position?
                IF INSTR("+-", Zeichen$) > 0 THEN
                  Zeichen$ = ""                    'Vorzeichen an erster
```

```
            END IF                                  'Position?
          END IF
        END IF
        IF Zeichen$<>"" THEN
          IF EditSpalte% <= MaxSpalte% THEN
            IF LEN(Eingab$) = Laenge% THEN
              Eingab$ = LEFT$(Eingab$, LEN(Eingab$) - 1)
            END IF
            Eingab$ = Eingab$ + Zeichen$
            LOCATE Zeile%, MinSpalte%
            PRINT Eingab$ + SPACE$(Laenge% - LEN(Eingab$));
            IF MaxSpalte% > EditSpalte% THEN
              EditSpalte% = EditSpalte% + 1
            END IF
          END IF
        END IF
      CASE ELSE
    END SELECT
  END IF
  IF Zeichen$ <> CHR$(13) THEN Zeichen$ = ""
WEND
END SUB

SUB Fenster (Zeile%, Spalte%, Breite%, Hoehe%, vfarbe%, hfarbe%) PUBLIC
  COLOR vfarbe%, hfarbe%
  LOCATE Zeile%, Spalte%
  PRINT "╒" + STRING$(Breite%, "=") + "╕";
  FOR x% = Zeile% + 1 TO Zeile% + Hoehe%
    LOCATE x%, Spalte%
    PRINT "│" + SPACE$(Breite%) + "│";
  NEXT x%
  LOCATE x%, Spalte%
  PRINT "╘" + STRING$(Breite%, "=") + "╛";
END SUB

SUB Hintergrund (vfarbe%, hfarbe%) PUBLIC
  COLOR vfarbe%, hfarbe%
  LOCATE 2,1
  FOR x% = 1 TO 23
    PRINT STRING$(80, "▒");
  NEXT x%
END SUB

SUB InfoZeile (Text$, Zeile%, vfarbe%, hfarbe%) PUBLIC
  LOCATE Zeile%, 1
  COLOR vfarbe%, hfarbe%
  Text$ = SollLen$(Text$, 80)
  PRINT Text$;
END SUB

FUNCTION SollLen$ (Text$, Laenge%) PUBLIC
  TextLaenge% = LEN(Text$)
  IF TextLaenge% > Laenge% THEN
      Text$ = LEFT$(Text$, Laenge%)
    ELSEIF TextLaenge% < Laenge% THEN
```

```
      Text$ = Text$ + SPACE$(Laenge% - TextLaenge%)
  END IF
  SollLen$ = Text$
END FUNCTION

SUB Warte PUBLIC
  WHILE INKEY$ = "": WEND
END SUB
```

Listing 2: TOOLS.BAS

```
$INCLUDE "TOOLS.H"            'Deklarationen der Unit TOOLS
$LINK "TOOLS.PBU"             'Benutzerbibliothek TOOLS einbinden

'*********************************************************************
'* Programmname              : HAUSHALT.BAS Version 1.00             *
'* Compiler                  : Power Basic 2.10                      *
'* verwendete Headerdateien: TOOLS.H                                 *
'* verwendete Units          : TOOLS.PBU                             *
'* geschrieben               : April/Mai 1991                        *
'* letzte Änderung           : 16.05.1991                            *
'*********************************************************************

$STACK 4000                   'Stapelspeicher erhöhen
CLS                           'Hauptprogramm HAUSHALT

ON ERROR GOTO FehlerBehandlung

vfarbe% = 7                   'Daten initialisieren
hfarbe% = 1
vErr%   = 14
hErr%   = 4
Anzahl% = 8                   'Menüeinträge
Menue$(1) = "Dateiwahl"
Menue$(2) = "Dateneingabe"
Menue$(3) = "Datenauswertung"
Menue$(4) = "Druck"
Menue$(5) = "Grafik"
Menue$(6) = "OS-Shell"
Menue$(7) = "Hilfe"
Menue$(8) = "Ende"

Start:
ArbeitsDatei$="keine"
CALL Hintergrund(vfarbe%, hfarbe%)
WHILE Wahl% <> 8
  CALL InfoZeile("HAUSHALT Version 1.00 - Einnahmen-/"+_
                 "Ausgabenüberschußrechnung", 1, vfarbe%, hfarbe%)
  CALL InfoZeile(CHR$(24)+CHR$(25)+": Anwahl Menüeintrag   RETURN:"+ _
                 "Menüauswahl    Arbeitsdatei: "+ ArbeitsDatei$ +_
                 Space$(12), 25, vfarbe%, hfarbe%)
  CALL Auswahl(Menue$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
  SELECT CASE Wahl%
    CASE 1   'Arbeitsdatei wählen und öffnen
```

```
      IF ArbeitsDatei$ <> "keine" THEN
        CLOSE #DNr
      END IF
      CALL DateiNamen(Namen$(), vfarbe%, hfarbe%, Datei%)
      ArbeitsDatei$ = Namen$(Datei%)
      DNr = FREEFILE
      OPEN ArbeitsDatei$ FOR RANDOM AS #DNr LEN=50
    CASE 2   'Dateneingabe
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenEingabe(DNr, vfarbe%, hfarbe%)
      END IF
    CASE 3   'Datenauswertung
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenAuswertung(1, DNr, vfarbe%, hfarbe%)
      END IF
    CASE 4    'Ergebnisse ausdrucken
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenAuswertung(2, DNr, vfarbe%, hfarbe%)
      END IF
    CASE 5    'Grafik
      IF ArbeitsDatei$="keine" THEN
          CALL KeineDateiAktiv(vfarbe%, hfarbe%)
        ELSE
          CALL DatenAuswertung(3, DNr, vfarbe%, hfarbe%)
      END IF
    CASE 6    'Tools (OS-Shell)
      COLOR 7,0
      CLS
      PRINT "zur Rückkehr zum Programm bitte EXIT eintasten..."
      SHELL
      CALL Hintergrund(vfarbe%, hfarbe%)
    CASE 7    'Hilfefunktion
      GOSUB Hilfe
    CASE ELSE
  END SELECT
WEND
CLOSE #DNr
COLOR 7, 0
CLS
PRINT "Programm beendet..."
END

'*********************************************************************
'* Fehlerbehandlung und Ereignisverfolgung                           *
'*********************************************************************

Fehlerbehandlung:
  SELECT CASE ERR
    CASE 6
```

```
      Fehler$="Überlauf!"
    CASE 14, 201
      Fehler$="unzureichender Speicherplatz!"
    CASE 24, 25, 68, 69
      Fehler$="Peripherie-, Schnittstellenprobleme!"
    CASE 27, 57
      Fehler$="Der Drucker ist nicht bereit!"
    CASE 51
      Fehler$="Interner Fehler!"
    CASE 70, 71, 72
      Fehler$="Probleme mit dem Laufwerk!"
    CASE 71
      Fehler$="Keine Diskette/Klappe offen!"
    CASE ELSE
      Fehler$="Fehler Nummer "+STR$(ERR)+" ist aufgetreten!"
  END SELECT
  CALL InfoZeile("Achtung: "+Fehler$, 25, vErr%, hErr%)
  CALL Warte
RESUME Start

Hilfe:
  CALL InfoZeile("Wählen Sie ein Thema, zu dem Sie Informationen"+_
                 " wünschen...", 25, hfarbe%, vfarbe%)
  CALL HinterGrund(vfarbe%, hfarbe%)
  Help$(1)="MENUE" : Help$(2)="DATEIWAHL"
  Help$(3)="DATENEINGABE" : Help$(4) = "AUSWERTUNG"
  Help$(5)="DRUCKER" : Help$(6)="GRAFIK"
  CALL Auswahl(Help$(), 6, hfarbe%, vfarbe%, X%)
  HilfeBegriff$=RTRIM$(Help$(X%))
  CALL Hintergrund(vfarbe%, hfarbe%)
  HNr = FREEFILE
  OPEN "HAUSHALT.HLP" FOR INPUT AS #HNr
  HAnz%=0
  WHILE NOT EOF(HNr)
    LINE INPUT #HNr, Zeile$
    IF INSTR(Zeile$, "["+HilfeBegriff$+"]") > 0 THEN
      WHILE Left$(Help$(HAnz%),1)<>"#"
        HAnz%=HAnz%+1
        LINE INPUT #HNr, Help$(HAnz%)
      WEND
    END IF
  WEND
  CALL Auswahl(Help$(), HAnz%-1, hfarbe%, vfarbe%, 0)
  CLOSE #HNr
  CALL Hintergrund(vfarbe%,hfarbe%)
RETURN

'*********************************************************************
'* Unterprogramme und Funktionen für das Programm HAUSHALT           *
'*********************************************************************
SUB DateiNamen(Namen$(), vfarbe%, hfarbe%, Wahl%)
  Wahl% = 0
  Anzahl% = 0                          'Dateieinträge ermitteln
  Datei$ = DIR$("*.DAT")               'Suchmaske festlegen
  WHILE Datei$<>"" AND Anzahl% < 10    'Einträge einlesen
```

```
    Anzahl% = Anzahl%+1
    Namen$(Anzahl%) = Datei$
    Datei$ = DIR$
  WEND
  a$ = ""
  IF Anzahl% = 0 THEN                    'keine Datei gefunden
      Namen$(1) = "HAUSHALT.DAT"         'Standardname erzeugen
      Wahl%= 1
    ELSEIF Anzahl%<10 THEN
      Anzahl% = Anzahl% + 1              'neue Datei erzeugen?
      Sonder$ = "neue Datei"
      Namen$(Anzahl%) = Sonder$
  END IF
  WHILE Wahl%=0                          'Arbeitsdatei ermitteln
    CALL Hintergrund (vfarbe%, hfarbe%)
    CALL InfoZeile("Info: Bitte wählen Sie eine Arbeitsdatei aus"+_
                   " dem folgenden Menü...", 1, vfarbe%, hfarbe%)
    ARRAY SORT Namen$() FOR Anzahl%      'Dateien alphabetisch sortieren
    CALL Auswahl (Namen$(), Anzahl%, vfarbe%, hfarbe%, Wahl%)
    IF RTRIM$(Namen$(Wahl%)) = Sonder$ THEN
      CALL Fenster(10, 30, 44, 1, vfarbe%, hfarbe%)
      Namen$(Wahl%) = RIGHT$(Date$, 4)
      CALL Eingabe("Bitte neuen Dateinamen eingeben: ", Namen$(Wahl%),_
                   8, 11, 32, vfarbe%, hfarbe%)
      IF RTRIM$(Namen$(Wahl%)) = "" THEN
        Namen$(Wahl%) = "HAUSHALT"
      END IF
      Namen$(Wahl%) = Namen$(Wahl%) + ".DAT"
    END IF
  WEND
  CALL Hintergrund (vfarbe%, hfarbe%)
END SUB

SUB KeineDateiAktiv(vfarbe%,hfarbe%)
  COLOR hfarbe%, vfarbe%
  CALL Fenster(10, 30, 48, 1, vfarbe%, hfarbe%)
  LOCATE 11, 32, 0
  PRINT "Bitte wählen Sie zunächst eine Arbeitsdatei..."
  CALL Warte
  COLOR vfarbe%, hfarbe%
END SUB

SUB DatenEingabe(DateiNummer, vfarbe%, hfarbe%)
  FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
        4 AS Betrag$
  SatzAnzahl& = LOF(DateiNummer) \ 50
  CALL Hintergrund(vfarbe%, hfarbe%)
  CALL Fenster(8, 10, 60, 8, vfarbe%, hfarbe%)
  CALL InfoZeile("PgUp/PgDn: Datensätzen blättern   Return: Buchung"+_
                 " vornehmen   "ESC: Eingabeende", 25, vfarbe%, hfarbe%)
  LOCATE 9, 12        'Eingabemaske ausgeben
  PRINT "Datensatznummer : ";
  LOCATE 11, 12
  PRINT "Buchungsdatum   : ";
  LOCATE 13, 12
```

```
PRINT "Beschreibung     : ["+SPACE$(36)+"]";
LOCATE 15, 12
PRINT "Einnahme/Ausgabe: ["+SPACE$(15)+"]";
CALL Fenster(19, 10, 60, 3, vfarbe%, hfarbe%)
LOCATE 20, 12
PRINT "Eine Einnahme wird mit einem positiven [+] und eine Ausgabe"
LOCATE 21, 12
PRINT "mit einem negativen Vorzeichen [-] in der ersten Spalte"
LOCATE 22, 12, 0
PRINT "gekennzeichnet. Die Verwendung des [+] ist dabei wahlfrei."
SatzNummer& = SatzAnzahl&
IF SatzNummer&>0 THEN
    GET #DateiNummer, SatzNummer&
    Datum$ = BuchungsDatum$
    Kommentar$ = Beschreibung$
    Ein$ = STR$(CVS(Betrag$))
  ELSE
    SatzNummer& = 1
    Datum$ = GermanDate$
END IF
EditFeld% = 0
CALL AktualisiereMaske (SatzNummer&, Datum$, Kommentar$, Ein$,_
                        vfarbe%, hfarbe%)
WHILE a$<>CHR$(27)
  a$ = INKEY$                        'ESC Eingabeende
  b$=RIGHT$(a$, 1)
  IF b$=CHR$(81) THEN                'PgUp (max. 500 000 Einträge)
      IF SatzNummer& < 500000 AND VAL(Ein$)<>0 THEN
        LSET BuchungsDatum$ = Datum$
        LSET Beschreibung$ = Kommentar$
        LSET Betrag$ = MKS$(VAL(Ein$))
        PUT #Dateinummer, SatzNummer&
        SatzNummer&=SatzNummer& + 1
        IF SatzNummer&>SatzAnzahl& THEN
            Datum$ = GermanDate$
            Kommentar$ = ""
            Ein$ = ""
            SatzAnzahl& = SatzAnzahl& + 1
          ELSE
            GET #Dateinummer, SatzNummer&
            Datum$ = BuchungsDatum$
            Kommentar$ = Beschreibung$
            Ein$ = STR$(CVS(Betrag$))
        END IF
      END IF
    ELSEIF b$=CHR$(73) THEN          'PgDn
      IF SatzNummer&>1 THEN
        LSET BuchungsDatum$ = Datum$
        LSET Beschreibung$ = Kommentar$
        LSET Betrag$ = MKS$(VAL(Ein$))
        PUT #Dateinummer, SatzNummer&
        SatzNummer&=SatzNummer& - 1
        GET #DateiNummer, SatzNummer&
        Datum$ = BuchungsDatum$
        Kommentar$ = Beschreibung$
```

```
          Ein$ = STR$(CVS(Betrag$))
        END IF
      ELSEIF a$=CHR$(13) THEN
        EditFeld% = EditFeld% XOR 1     'editiere 1. bzw. 2.Feld
        IF EditFeld%=1 THEN
            Kommentar$ = RTRIM$(Kommentar$)
            CALL Eingabe("", Kommentar$, 36, 13, 31, vfarbe%, hfarbe%)
          ELSE
            Ein$ = RTRIM$(Ein$)
            CALL Eingabe("R|", Ein$, 15, 15, 31, vfarbe%, hfarbe%)
        END IF
    END IF
    IF b$=CHR$(73) OR b$=CHR$(81) OR a$=CHR$(13) THEN
      CALL AktualisiereMaske (SatzNummer&, Datum$, Kommentar$, Ein$,_
                              vfarbe%, hfarbe%)
    END IF
  WEND
  CALL HinterGrund(vfarbe%, hfarbe%)
END SUB

FUNCTION GermanDate$
  GermanDate$=MID$(Date$,4,2) + "." + LEFT$(Date$,2) + "." +_
              RIGHT$(Date$,4)
END FUNCTION

SUB AktualisiereMaske(SatzNummer&, Datum$, Kommentar$, Wert$, vfarbe%,_
                      hfarbe%)
  COLOR vfarbe%, hfarbe%
  LOCATE  9, 31, 0 : PRINT SatzNummer&; SPACE$(10)
  LOCATE 11, 31 : PRINT Datum$
  LOCATE 13, 31 : PRINT SollLen$(Kommentar$, 36)
  LOCATE 15, 31 : PRINT SollLen$(Wert$,15)
END SUB

SUB DatenAuswertung(Mode%, Dateinummer, vfarbe%, hfarbe%)
  FIELD #Dateinummer, 10 AS BuchungsDatum$, 36 AS Beschreibung$,_
        4 AS Betrag$
  CALL Hintergrund(vfarbe%, hfarbe%)
  CALL Fenster(10, 16, 46, 1, vfarbe%, hfarbe%)
  Jahr$ = RTRIM$(RIGHT$(Date$, 4))
  IF Mode%=1 THEN
      Teil$ = "Bildschirm"
    ELSEIF Mode%=2 THEN
      Teil$ = "Drucker"
    ELSE
      Teil$ = "Grafik"
  END IF
  CALL Eingabe("I|"+ Teil$+"auswertung für welches Jahr: ", Jahr$, 4, 11,_
               18, vfarbe%, hfarbe%)
  IF LEN(Jahr$)<> 4 THEN EXIT SUB
  LOCATE ,,0
  FOR SatzNummer&=1 TO LOF(Dateinummer)\50
    GET #Dateinummer, SatzNummer&
    Datum$ = BuchungsDatum$
    Kommentar$ = Beschreibung$
```

```
  Betrag! = CVS(Betrag$)
  IF RIGHT$(Datum$, 4) = Jahr$ THEN
    IF Betrag! < 0 THEN                          'Gesamtsummen
        Ausgabe! = Ausgabe! + ABS(Betrag!)
      ELSEIF Betrag! > 0 THEN
        Einnahme! = Einnahme! + Betrag!
    END IF
    IF Mode%=2 THEN                              'Druckerausgabe
      Summe! = Einnahme! - Ausgabe!
      CALL DruckerAusgabe(SatzNummer&, Datum$, Kommentar$, Betrag!,_
                          Summe!)
    END IF
    SELECT CASE MID$(Datum$, 4, 2)               'monatlicher Saldo
      CASE "01"   'Januar
        Januar! = Januar! + Betrag!
      CASE "02"   'Februar
        Februar! = Februar! + Betrag!
      CASE "03"   'März
        Maerz! = Maerz! + Betrag!
      CASE "04"   'April
        April! = April! + Betrag!
      CASE "05"   'Mai
        Mai! = Mai! + Betrag!
      CASE "06"   'Juni
        Juni! = Juni! + Betrag!
      CASE "07"   'Juli
        Juli! = Juli! + Betrag!
      CASE "08"   'August
        August! = August! + Betrag!
      CASE "09"   'September
        September! = September! + Betrag!
      CASE "10"   'Oktober
        Oktober! = Oktober! + Betrag!
      CASE "11"   'November
        November! = November! + Betrag!
      CASE "12"   'Dezember
        Dezember! = Dezember! + Betrag!
      CASE ELSE
    END SELECT
  END IF
NEXT SatzNummer&
Gesamt! = Einnahme! - Ausgabe!
IF Mode% = 1 THEN
    CALL Fenster(5, 5, 70, 15, vfarbe%, hfarbe%)
    Gesamt! = Einnahme! - Ausgabe!
    LOCATE  6, 8: PRINT "Auswertungsjahr.........:  ";Jahr$
    LOCATE  7, 8: PRINT "Summe aller Einnahmen...: ";Runden!(Einnahme!)
    LOCATE  8, 8: PRINT "Summe aller Ausgaben....: ";Runden!(Ausgabe!)
    LOCATE  9, 8: PRINT "Gesamtsaldo.............: ";Runden!(Gesamt!)
    LOCATE 10, 5: PRINT "├" + STRING$(70, "-") + "┤";
    LOCATE 12, 8: PRINT "Saldo Januar...:";Runden!(Januar!)
    LOCATE 12,45: PRINT "Juli...........:";Runden!(Juli!)
    LOCATE 13, 8: PRINT "      Februar..:";Runden!(Februar!)
    LOCATE 13,45: PRINT "August.........:";Runden!(August!)
    LOCATE 14, 8: PRINT "      März.....:";Runden!(Maerz!)
```

```
      LOCATE 14,45: PRINT "September......:";Runden!(September!)
      LOCATE 15, 8: PRINT "      April....:";Runden!(April!)
      LOCATE 15,45: PRINT "Oktober........:";Runden!(Oktober!)
      LOCATE 16, 8: PRINT "      Mai......:";Runden!(Mai!)
      LOCATE 16,45: PRINT "November.......:";Runden!(November!)
      LOCATE 17, 8: PRINT "      Juni.....:";Runden!(Juni!)
      LOCATE 17,45: PRINT "Dezember.......:";Runden!(Dezember!)
      LOCATE ,,0
      CALL Warte
    ELSEIF Mode% = 2 THEN
      LPRINT : LPRINT
      LPRINT Space$(3)+ "Auswertungsjahr.........:  ";Jahr$
      LPRINT Space$(3)+ "Summe aller Einnahmen...: ";Runden!(Einnahme!)
      LPRINT Space$(3)+ "Summe aller Ausgaben....: ";Runden!(Ausgabe!)
      LPRINT Space$(3)+ "Gesamtsaldo.............: ";Runden!(Summe!)
      CALL FormFeed
    ELSEIF Mode% = 3 THEN            'Kreisdiagramm
      SCREEN 12                      'Grafikmodus
      CALL Linien(Januar!,Februar!, Maerz!, April!, Mai!, Juni!, Juli!,_
           August!, September!, Oktober!, November!, Dezember!)
      CLS
      IF Einnahme!<>0 AND Ausgabe!<>0 THEN
        CALL Kreis(Einnahme!, Ausgabe!)
      END IF
      SCREEN 0                       'zurück in Textmodus
  END IF
  CALL Hintergrund(vfarbe%, hfarbe%)
END SUB

FUNCTION Runden!(Zahl!)
  Runden! = INT(Zahl! * 100 + 0.5) / 100
END FUNCTION

SUB FormFeed
  LPRINT CHR$(12);
END SUB

SUB SeitenKopf(Seite%, ZeilenZahl%)
  Seite% = Seite% + 1
  LPRINT
  LPRINT SPACE$(3) + "HAUSHALT Version 1.00  - Druckerausgabe"+_
         " Datenauswertung -  Seite: ";Seite%
  LPRINT
  LPRINT
  LPRINT SPACE$(3) + "Nr.   |Buchung   |Kommentar"+_
         "                          |Buchung  |Saldo     "
  LPRINT SPACE$(3) + "──────┼──────────┼"+_
                     "──────────────────────────────────┼"+_
                     "──────┼──────────"
  ZeilenZahl% = 6
END SUB

SUB DruckZeile(Nr&, Datum$, Kommentar$, Betrag!, Summe!)
  Nr$ = SollLen$(LTRIM$(STR$(Nr&)), 6)
  Betrag$ = LTRIM$(STR$(Betrag!))
```

```
  Summe$ = LTRIM$(STR$(Summe!))
  LPRINT SPACE$(3) + Nr$ +  "|" + Datum$ +  "|" + Kommentar$ +  "|" +_
         Formatiere$(Betrag$, 9) +  "|" + Formatiere$(Summe$, 10)
END SUB

SUB DruckerAusgabe(SatzNummer&, Datum$, Kommentar$, Betrag!, Summe!)
 STATIC ZeilenZahl%, Seite%
 IF ZeilenZahl% = 0 OR ZeilenZahl% = 60 THEN
   CALL SeitenKopf(Seite%, ZeilenAnzahl%)
 END IF
 CALL DruckZeile(SatzNummer&, Datum$, Kommentar$, Runden!(Betrag!),_
                 Runden!(Summe!))
 ZeilenZahl% = ZeilenZahl% + 1
 IF ZeilenZahl% = 60 THEN
   CALL FormFeed
   ZeilenZahl% = 0
 END IF
END SUB

FUNCTION Formatiere$(Zahl$, Laenge%)
  Komma% = INSTR(Zahl$, ".")
  IF Komma% = 0 THEN
      Zahl$ = Zahl$ + ".00"
    ELSEIF INSTR(Zahl$, ".") = LEN(Zahl$)-1 THEN
      Zahl$ = Zahl$ + "0"
  END IF
  IF Laenge%> LEN(Zahl$) THEN
    Zahl$ = SPACE$(Laenge%-LEN(Zahl$)) + Zahl$
  END IF
  Formatiere$ = Zahl$
END FUNCTION

SUB Kreis(Einnahme!, Ausgabe!)
  maxX% = 640-1                          'Grafikmodus
  maxY% = 480-1
  G! = Ausgabe!/(Einnahme!/360)
  LOCATE 30,1                            'Text / Untergrund
  COLOR 1, 3
  PRINT "Fortsetzung mit beliebiger Taste...";
  COLOR 14                               'Beschriftung
  LOCATE 3, 20
  PRINT "Kreisdiagramm Einnahmen/Ausgaben Gesamt in DM"
  LOCATE 4, 20
  PRINT "Gesamteinnahmen: ";Runden!(Einnahme!);"DM"
  COLOR 5
  LOCATE 15, 61
  PRINT "Saldo   =";Runden!(Einnahme!-Ausgabe!)
  COLOR 1
  LOCATE 16, 61
  PRINT "Ausgaben=";Runden!(Ausgabe!)
  LINE (0, 0) - (maxX%, maxY%), 15, B     'Rahmen
  LINE (0, maxY%-16) - (maxX%, maxY%-16), 15
  CIRCLE (maxX%\2, maxY%\2), 150, 15      'Kreisdiagramm
  PAINT (maxX%\2, maxY%\2), 5, 15
  CIRCLE (maxX%\2, maxY%\2), 150, 15, -0.001, (-1) * Grad2Rad#(G!)
```

```
  PAINT (maxX%\2+10, maxY%\2+2), 2,15
  CALL Warte                          'warten auf Tastendruck
END SUB

SUB Linien(Jan!,Feb!,Mar!,Apr!,Mai!,Jun!,Jul!,Aug!,Sep!,Okt!,Nov!,Dez!)
  maxX% = 640-1                       'Grafikmodus
  maxY% = 480-1
  LOCATE 30,1                         'Text / Untergrund
  COLOR 1, 3
  PRINT "Fortsetzung mit beliebiger Taste...";
  COLOR 14                            'Beschriftung
  LOCATE 3, 25
  PRINT "Salden bezogen auf einzelne Monate"
  LOCATE 4, 20
  LINE (0, 0) - (maxX%, maxY%), 15, B     'Rahmen
  LINE (0, maxY%-16) - (maxX%, maxY%-16), 15
  LINE (10, maxY%\2) - (maxX%-10, maxY%\2), 1
  DRAW "C1 H10 F10 G10"               'Pfeil rechts
  LINE (20, maxY%-30) - (20, 35), 1
  DRAW "C1 G10 E10 F10"               'Pfeil rechts
  DRAW "BM 10,"+STR$(maxY%\2)         'Ursprung
  FOR x%=1 TO 12                      'x-Koordinatenleiste
    DRAW "R45 D5 U5"
  NEXT x%
  COLOR 1
  LOCATE  2, 3: PRINT "DM*1000"
  LOCATE 16, 2: PRINT "0"
  LOCATE 16,72: PRINT "Monat"
  LOCATE 17, 6: PRINT "Jan"
  LOCATE 17,24: PRINT "Apr"
  LOCATE 17,46: PRINT "Aug"
  LOCATE 17,68: PRINT "Dez"
  DRAW "BM 20,"+STR$(maxY%\2)         'y-Koordinatenleiste
  FOR x%=1 TO 3
    DRAW "D60 R5 L5"
  NEXT x%
  DRAW "BM 20,"+STR$(maxY%\2)         'y-Koordinatenleiste
  FOR x%=1 TO 3
    DRAW "U60 R5 L5"
  NEXT x%
  LOCATE  4,5: PRINT "+15"
  LOCATE  8,5: PRINT "+10"
  LOCATE 12,5: PRINT "+5"
  LOCATE 19,5: PRINT "-5"
  LOCATE 23,5: PRINT "-10"
  LOCATE 27,5: PRINT "-15"            'Liniengrafik
  DRAW "BM 55,"+STR$((maxY%\2)-Pixelwert%(Jan!))
  xWert%=55
  FOR x% = 2 TO 12
    SELECT CASE x%
      CASE  2: Monat! = Feb!
      CASE  3: Monat! = Mar!
      CASE  4: Monat! = Apr!
      CASE  5: Monat! = Mai!
      CASE  6: Monat! = Jun!
```

```
      CASE  7: Monat! = Jul!
      CASE  8: Monat! = Aug!
      CASE  9: Monat! = Sep!
      CASE 10: Monat! = Okt!
      CASE 11: Monat! = Nov!
      CASE 12: Monat! = Dez!
      CASE ELSE
    END SELECT
    xWert% = xWert%+45
    LINE -(xWert%,(maxY%\2)-PixelWert%(Monat!)), 14
  NEXT x%
  CALL Warte
END SUB

FUNCTION Grad2Rad#(G!)
  pi# = 3.141592654
  Grad2Rad# = ((2 * pi#)/360) * G!
END FUNCTION

FUNCTION PixelWert%(Wert!)
  PixelWert% = INT(Wert! * (60/5000))
END FUNCTION
```

Listing 3: HAUSHALT.BAS

B Glossar

Anwendungsprogramm: Programm zum Lösen von Benutzerproblemen.

Cursor: Zeichen, das Ihnen auf dem Bildschirm anzeigt, an welcher Stelle Sie Eingaben vornehmen können.

Compiler: Programm, das einen Quelltext einer höheren Programmiersprache in Maschinensprache übersetzt und Sprungmarken für die Bibliotheksroutinen einfügt.

Coprozessor: Hardware-Zusatz, der mathematische Fließkommaberechnungen beschleunigt.

Debugger: Programm, mit dem die Fehlersuche in Programmen ermöglicht wird.

DOS: Software, die den Betrieb eines Rechners ermöglicht (**D**isk **O**perating **S**ystem).

Editor: Programm, das die Bearbeitung eines ASCII-Textes erlaubt.

Grafikkarten: Hardware-Zusätze, die die grafische Bildverarbeitung auf einem Computer ermöglichen.

GW-Basic/BasicA: Basic-Interpreter, der mit dem Betriebssystem MS-DOS bzw. PC-DOS von 1981 bis 1991 ausgeliefert wurde.

Hardware: Als Hardware bezeichnet man die technischen Geräte eines Computersystems.

IDE: Integrierte Entwicklungsumgebung, die mehrere Programme unter einer einheitlichen Benutzeroberfläche vereinigt. In Power Basic sind der Editor, Compiler, Linker und Debugger unter einer einheitlichen Oberfläche zusammengefaßt.

Interpreter: Ein Programm, das Programmcode einliest und schrittweise ausführt.

kompatibel: Meint im Computerbereich soviel wie »austauschbar«.

Linker: Programm, das die vom Compiler gesetzten Sprungmarken durch Einbinden von Bibliotheksroutinen auflöst.

Maschinencode: Anweisungen, die vom Prozessor verarbeitbar sind. Eine Programmiersprache dient als »Dolmetscher« und übersetzt die Befehle höherer Programmiersprachen in ein für den Rechner verständliches Maschinenprogramm.

Programm: Eine exakte Darstellung eines Algorithmus bezeichnet man als Programm. Ein Algorithmus ist die Beschreibung eines Lösungsverfahrens für eine Klasse von Problemen.

Programmiersprache: Sprache, mit der Algorithmen und Programme für einen Computer formuliert werden können.

Unit: Bei einer Unit handelt es sich um eine Funktionsbibliothek, in der benutzerdefinierte Funktionen und Unterprogramme in Maschinencode zusammengefaßt sind.

Quelltext: Ein Programm, das im ASCII-Format mit einem Editor erstellt wurde.

Software: Gesamtheit aller Programme, die auf einem Rechner eingesetzt werden können.

Syntax: Die Syntax bestimmt die sprachliche Notierung von Programmen. Bei den natürlichen Sprachen, wie z.B. Englisch oder Deutsch, bezeichnet man die Syntax als »Grammatik«.

C Tabellen

In der ersten Tabelle sind die wichtigsten Tastaturabfragecodes, die Sie beim Einsatz der *INKEY$*-Anweisung benötigen, zusammengestellt. Dabei ist berücksichtigt, daß viele Tasten in Verbindung mit Sondertasten, wie der [Alt]- oder [Strg]-Taste, unterschiedliche Codes zurückliefern. Der Code selber ist als Dezimalwert dargestellt. Da Sie mit *INKEY$* immer eine Zeichenkette erhalten, könnten Sie mit der *CHR$*-Funktion den Dezimalwert in ein Zeichen konvertieren (siehe Routinen »Auswahl und Eingabe in der Funktionsbibliothek TOOLS.BAS«), damit ein Vergleich möglich ist.

Ferner sind in einer weiteren Tabelle die Zeichen aus dem ASCII-Zeichensatz zusammengestellt, die Sie z.B. zum Erstellen von Rahmen auf dem Textbildschirm oder dem Drucker benötigen.

Tastaturabfragecodes

Beachten Sie, daß, je nach gedrückter Tastenkombination, ein Ein- bzw. Zwei-Zeichen-Tastaturcode zurückgeliefert wird. Besteht die Information aus einem Zeichen, kann der Wert direkt über *INKEY$* ermittelt werden. Werden zwei Zeichen zurückgegeben, ist das erste Zeichen immer ein ASCII 0. In der Tabelle erscheint in einem solchen Fall die Codefolge *0+ZweiterZeichenCode*.

```
'einfache Auswertung
Zeichen$=""
WHILE Zeichen$=""
  Zeichen$=INKEY$
WEND
IF Zeichen$=CHR$(27) THEN
   'Esc-Taste gedrückt
END IF
```

Mit den nachfolgenden Anweisungen können Sie ermitteln, ob eine Taste bzw. eine Tastenkombination einen Ein- bzw. Zwei-Zeichen-Tastaturcode zurückliefert.

```
'ermitteln ob Zwei-Zeichen-Tastaturcode
Zeichen$=""
WHILE Zeichen$<>""
  Zeichen$=INKEY$
WEND
IF LEN(Zeichen$)=2 THEN
    'Zwei-Zeichen-Tastaturcode
    'Information im rechten Teil
    ZeichenInfo$=RIGHT$(Zeichen$,1)
  ELSE
    'Ein-Zeichen-Tastaturcode
END IF
```

Taste	Code	Shift	Strg	Alt
Esc	27	27	27	
Backspace	8	8	127	
Tab	15	0+15		
Return	13	13	10	
F1	0+59	0+84	0+94	0+104
F2	0+60	0+85	0+95	0+105
F3	0+61	0+86	0+96	0+106
F4	0+62	0+87	0+97	0+107
F5	0+63	0+88	0+98	0+108
F6	0+64	0+89	0+99	0+109
F7	0+65	0+90	0+100	0+110
F8	0+66	0+91	0+101	0+111
F9	0+67	0+92	0+102	0+112
F10	0+68	0+93	0+103	0+113
Home/Posl	0+71			
↑	0+72			
PgUp	0+73			
←	0+75			
→	0+77			
End	0+79			
↓	0+80			
PgDn	0+81			
Ins/Einfg	0+82			
Del/Entf	0+83			

ASCII-Tabelle

☺	1	☻	2	♥	3	♦	4	♣	5	♠	6	•	7	◘	8	○	9	◙	10	♂	11
♀	12	♪	13	♫	14	☼	15	►	16	◄	17	↕	18	‼	19	¶	20	§	21	▬	22
↨	23	↑	24	↓	25	→	26	←	27	∟	28	↔	29	▲	30	▼	31		32	!	33
"	34	#	35	$	36	%	37	&	38	'	39	(| 40 |)	41	*	42	+	43	,	44
-	45	.	46	/	47	0	48	1	49	2	50	3	51	4	52	5	53	6	54	7	55
8	56	9	57	:	58	;	59	<	60	=	61	>	62	?	63	@	64	A	65	B	66
C	67	D	68	E	69	F	70	G	71	H	72	I	73	J	74	K	75	L	76	M	77
N	78	O	79	P	80	Q	81	R	82	S	83	T	84	U	85	V	86	W	87	X	88
Y	89	Z	90	[	91	\	92	]	93	^	94	_	95	`	96	a	97	b	98	c	99
d	100	e	101	f	102	g	103	h	104	i	105	j	106	k	107	l	108	m	109	n	110
o	111	p	112	q	113	r	114	s	115	t	116	u	117	v	118	w	119	x	120	y	121
z	122	{	123	|	124	}	125	~	126	⌂	127	Ç	128	ü	129	é	130	â	131	ä	132
à	133	å	134	ç	135	ê	136	ë	137	è	138	ï	139	î	140	ì	141	Ä	142	Å	143
É	144	æ	145	Æ	146	ô	147	ö	148	ò	149	û	150	ù	151	ÿ	152	Ö	153	Ü	154
¢	155	£	156	¥	157	₧	158	ƒ	159	á	160	í	161	ó	162	ú	163	ñ	164	Ñ	165
ª	166	º	167	¿	168	⌐	169	¬	170	½	171	¼	172	¡	173	«	174	»	175	░	176
▒	177	▓	178	│	179	┤	180	╡	181	╢	182	╖	183	╕	184	╣	185	║	186	╗	187
╝	188	╜	189	╛	190	┐	191	└	192	┴	193	┬	194	├	195	─	196	┼	197	╞	198
╟	199	╚	200	╔	201	╩	202	╦	203	╠	204	═	205	╬	206	╧	207	╨	208	╤	209
╥	210	╙	211	╘	212	╒	213	╓	214	╫	215	╪	216	┘	217	┌	218	█	219	▄	220
▌	221	▐	222	▀	223	α	224	ß	225	Γ	226	π	227	Σ	228	σ	229	µ	230	τ	231
Φ	232	Θ	233	Ω	234	δ	235	∞	236	ø	237	∈	238	∩	239	≡	240	±	241	≥	242
≤	243	⌠	244	⌡	245	÷	246	≈	247	°	248	∙	249	·	250	√	251	ⁿ	252	²	253
■	254		255																		

D Literaturhinweis

Power Basic befindet sich erst seit kurzer Zeit auf dem Markt, so daß leider bislang keine spezielle Sekundärliteratur erhältlich ist. Lassen sich Toolboxen anderer Basic-Dialekte nicht ohne weiteres verwenden, kann man sich Anregungen und Lösungen dennoch aus bereits vorhandener Basic-Literatur holen. Dabei sollte man Literatur zu Turbo Basic unbedingt der Literatur zu Quick Basic vorziehen.

Eine Zeitschrift, die sich regelmäßig mit Themen zu Power Basic befaßt und spezielle Programmierkurse im Heft anbietet, ist die Computer-Zeitschrift »Highscreen Highlights«, die im DMV-Verlag erscheint und derzeit exklusiv bei VOBIS vertrieben wird. Ein Bezug dieser Fachzeitschrift soll ab September auch über den Zeitschriftenhandel möglich sein.

Stichwortverzeichnis

$INCLUDE 70
$LINK 35, 49, 70
$SEGMENT 35
$STACK 49
$STRING 92

A

Altgrad 122
APPEND 139
Arbeitsdiskette anlegen 16
Arbeitsfläche definieren 37
Array 26
ASC() 81
ASCII-Tabelle 264
ASCII-Zeichensatz 39
Ausdrücke 26
Ausführbare Programmdatei 152
Auswahlroutine 42

B

BAK-Datei 165
BCD-Festzahl 25
BCD-Fließzahl 25
Benutzerbibliothek 33
Benutzerhandbuch 12
Benutzeroberfläche 22
Betriebssystem 16, 57, 142
Bildschirmauflösung 118
Bildschirmausgaben 67
Bildschirmmodi 114
Binärdatei 55
Blockbefehle 47
Bogenmaß 89, 122
Breakpoint 161

C

CALL 31
CHR$() 81
CIRCLE 118, 122, 124
CLOSE 54
COM-Datei 152
Compiler 11, 152, 260
Compiler-Anweisungen 190
Compiler-Befehle 49
Control-Break 153
Coprozessor 12, 170, 260
Cursor 260
Cursorsteuerung 21

D

DATE$ 82
Datei 54, 195
Dateiarten 55
Dateiauswahl 61
Dateikürzel 57
Dateimanager 54
Dateiname 57
Dateinummer 55
Dateipuffer 81
Dateiverwaltung 83
Dateiverzeichnis 56
Daten aufbereiten 103
Datenauswertung 93
Dateneingabe 75

Datenfeld 26, 68
Datenfelder verwalten 56
Datenfeldoperationen 199
Datensatz 68, 75
Datensicherung 164
Datentypen 24, 80
Debugger 160, 260
Deklarationsteil 41, 44
Dienstprogramm 19
DIM 26
Direktzugriffsdatei 55, 67, 81
Disketteninstallation 18
DO-LOOP-Schleife 27
DOS 260
DOS-Anweisungen 54, 201
DOS-Piping 165
DRAW 116, 125
Drucker-Steuercode 104
Druckerausgabe 101, 105
Druckersteuerung 204
dynamische Zeichenkette 92

E

Editor 19, 260
EGA 114
Eingabefenster 15
Eingabemaske erstellen 67
Eingaberoutine 36
Einzelschrittmodus 160
End-Of-File-Marke 140
Entscheidungskonstrukte 29
Entwicklungsumgebung 11, 163
EOF 93, 140
Epson-Drucker 104
Epson-Emulation 104
Ereignisverfolgung 141
ERR 136
EXE-Datei 152

F

Farbeinstellungen 19
Fehlerbehandlung 134, 159, 210
Fehlerbeseitigung 157
Fehlermeldungen 157
Fensterverwaltung 39, 169
Festplatteninstallation 18
FIELD 68, 79
Fließkommavariable 26
FOR-NEXT-Schleife 27, 38
FREEFILE 61, 148
Funktionen 30, 36
Funktionsbibliothek 173

G

GET 82
Glossar 260
GOTO 27
Grafik 211
Grafikkarte 260
Grafikmodi 113
Grafikmodus 113, 122
Grafikprogrammierung 113
Grafikroutinen 115
GW-Basic 12, 260
GW-Basic-Programme übernehmen 167

H

Haltepunkt 161
Hardware-Anforderungen 16
Hilfe-Index 140
Hilfsdatei erstellen 140

I

IDE 260
INKEY$ 42, 73
INPUT 67, 139
INPUT# 139
Installation 16
Diskette 17
Festplatte 17
Integerkonstante 26
Integervariable 24
Interpreter 11, 260
Interrupt-Programmierung 166

K

Kommandozeilen-Compiler 171
Kommentar 48
Konstante 26
kontextsensitives Hilfesystem 140
Kontrollanweisung 27
Kontrollstruktur 27
Konvertierfunktionen 213
Konvertierroutine 80
Koordinatensystem 118
Kreisdiagramm 118, 121

L

Laufzeitbibliothek 152
Laufzeitfehler 68, 120, 129, 132, 157
LCASE$ 92
Lieferumfang 17
LINE 116, 118, 125
LINE-INPUT# 139
Liniendiagramm 118
Linker 152, 154, 260
Literaturhinweis 265
LOCATE 99
LOF 82, 140
Long-Integer 25
LPRINT 101
LSET 81
LTRIM$ 110

M

Makrosprache 116
Maschinencode 261
Maschinenprogramm 155
Maschinensprache 152
mathematische Funktionen 90, 219
Mausunterstützung 15
MCGA 114
Menübefehle 15
Menüreferenz 175
Modulare Programmierung 36
Musik 222

O

ON-ERROR 131
OPEN 54, 76, 139
Operatoren 26, 89
OUTPUT 139

P

PAINT 124
parallele Schnittstelle 101
Parameterliste 71
PBINST 19
Pfeiltasten 21
Pick-Liste 70
Pixelgrafik 116
Power Basic konfigurieren 19
Power-Basic-Datentypen 25
Power-Basic-Grafikmodi 114
Programmablauf 27
Programmablaufsteuerung 222
Programmaufbau 41
Programm-Mängel 164
Programmzähler 159
PUBLIC 48
PUT 81

Q

Quad-Integer 25
Quelltext 261
Quelltext-Debugger 160

R

Randomdatei 55, 68, 79, 86
Referenzhandbuch 12
RESUME 132
RSET 81

S

Schleifen 27
Schleifenvariable 28

Schrittweite 28
SCREEN 114, 120
SELECT-CASE-Anweisung 29
sequentielle Datei 55, 93
sequentielle Dateiverwaltung 138
SHELL 142, 165
Sicherungsdatei 165
Sicherungskopie 17
Software-Anforderungen 16
SPACE$ 110
Speicherverwaltung 229
Sprachreferenz 189
Sprachsyntax 24, 32
Sprungmarke 136
STATIC 111
STEP 28
Steuercode 104
STR$ 81, 110
Stringfunktionen 91
Stringoperationen 232
Strukturierte Programmierung 35
SUB 23, 38
Suffix 57
Syntax 261
Syntaxfehler 129, 157
Systemabsturz 164, 172
Systemdatum 82
Systemprompt 17
Systemzeit 82

T

Tabellenausdruck 111
Tastaturabfrage 33
Tastaturabfragecodes 262
Tastaturbelegung 19
Tastaturcode 43
Testphase 151
Textformatierung 235
TIME$ 82
Toolboxen 172
Typkonvertierung 81
Typzuweisung 26

U

UCASE$ 92
Unit 34, 47, 261
Unterprogramme 30, 36, 38

V

VAL 81
Variablendeklarationen 25, 193
Variablenmanipulation 163
Variablenname 25
Vektorgrafik 116
VGA 113, 170

W

Wertebereich 25
WHILE-WEND-Schleife 28
WIDTH 102
WINDOW 118

X

X/Y-Koordinatensystem 118
XOR 77

Z

Zeichenkette 26
Zeichenprogramm 116
Zeilennummer 24
Zusatz-Software 172

Befehlskarte

Tastenbefehle des Power-Basic-Editors

Strg + Q E	Fensteranfang
Strg + Q X	Fensterende
Strg + Q R	Dateianfang
Strg + Q C	Dateiende
Strg + Q B	Blockanfang
Strg + Q K	Blockende
Strg + Q P	letzte Cursorposition
Strg + V	Einfüge-/Überschreibmodus
Strg + H	Zeichen links vom Cursor löschen
Strg + G	Zeichen unter Cursor löschen
Strg + T	Wort hinter Cursor löschen
Strg + N	neue Zeile einfügen
Strg + Y	Zeile löschen
Strg + Q Y	bis Zeilenende löschen
Strg + U	Operation abbrechen
Strg + O I	Einrückung Ein/Aus
Strg + P	Control-Zeichen-Eingabe
Strg + O F	Füllmodus
Strg + Q n	Suche Markierung n = 1, 2 oder 3
Strg + K D	Editor verlassen ohne zu speichern
Strg + K Q	dto.
Strg + L	letzte Suche wiederholen
Strg + Q W	wiederhole Fehlermeldung
Strg + Q L	Zeilenänderung rückgängig machen
Strg + K S	Datei speichern und fortsetzen
Strg + Q F	suchen
Strg + Q A	suchen und ersetzen
Strg + K n	Markierung setzen, n = 1, 2 oder 3
Strg + I	Tabulator
Strg + O T	Tabulator-Modus
F10	Menüauswahl
Strg + O U	Einrückungsmodus

Power-Basic-Funktionstastenbelegung

F1	Kontextsensitive Hilfsfunktion aktivieren
F2	Datei auf Festplatte/Diskette speichern
F3	Datei von Festplatte/Diskette in den Editor laden
F5	Aktives Fenster vergrößern
F6	Umschaltung zwischen Edit- und Watch-Fenster
F7	Unterprogramme bzw. Funktionen schrittweise ausführen
F8	Unterprogramme/Funktionen in einem Schritt ausführen
F9	Programm starten
F10	Umschaltung zwischen Menü und aktivem Fenster

Tastenbefehle des Power-Basic-Editors

Strg + K B	Blockanfang markieren
Strg + K K	Blockende markieren
Strg + K H	Blockmarkierung entfernen
Strg + K C	Block an Cursorposition kopieren
Strg + K Y	Block löschen
Strg + K V	Block bewegen
Strg + K P	Programm oder Block drucken
Strg + K T	einzelnes Wort markieren
Strg + K R	Block von Diskette einlesen
Strg + K W	Block auf Diskette schreiben
Esc	Editor verlassen
Strg + S	ein Zeichen nach links
Strg + D	ein Zeichen nach rechts
Strg + A	ein Wort nach links
Strg + F	ein Wort nach rechts
Strg + E	eine Zeile nach oben
Strg + X	eine Zeile nach unten
Strg + W	eine Zeile hochscrollen
Strg + Z	eine Zeile runterscrollen
Strg + R	eine Seite nach oben
Strg + C	eine Seite nach unten
Strg + Q S	Zeilenanfang
Strg + Q D	Zeilenende

Markt&Technik **W O R K S H O P PC**

Erfolgreich starten – sicher nutzen.

Power Basic

Zum Programm

Power Basic ist ähnlich leistungsfähig wie C oder Pascal, von denen viele Sprachelemente übernommen wurden. Das bedeutet klare Programmstrukturen, hohe Geschwindigkeit und hoher Bedienungskomfort.
Allerdings erschwert die Fülle der Befehle den leichten Einstieg. Hier bietet der Autor Hilfe.

Zum Buch

Workshop – das heißt: sichere Lernerfolge in kürzester Zeit.

Erfolgreich starten

Im Einführungskapitel erfahren Sie alles, was zum grundlegenden Umgang mit Power Basic und dem Buch notwendig ist. Außerdem: Starthilfe in Sachen Installation. Das Tutorium führt Sie dann in zehn Sitzungen in diese leistungsfähige Programmiersprache ein – ausführlich und praxisgerecht –, anhand eines nützlichen wie auch vielfältigen Projekts. Das Ergebnis: Eine Einnahmen-/Ausgabenverwaltung mit moderner grafischer Bedienoberfläche. Die Routinen für Dateiverwaltung, Grafik oder Menüführung lassen sich jederzeit in eigene Programme einbinden.

Sicher nutzen

Das Kapitel Know-how wird Ihr Ratgeber für die tägliche Praxis: häufige Fehlerquellen und entsprechende Lösungsvorschläge; eine umfangreiche Sammlung nützlicher Tips und Kniffe und vieles mehr. Dann die Referenz: Knapp und präzise – alle Befehle auf einen Blick. Als nützliche Zugabe: Eine handliche Befehlskarte mit dem Allerwichtigsten. Der Abschnitt Schnellinformation bietet schließlich nach dem Motto »Wie programmiere ich ...?« eine Reihe von ausgewählten Tools zu häufigen Problemen in der Praxis.

Workshop – verständlich und informativ; zu populären Themen wie Grafik, Musik, Textverarbeitung und Programmiersprachen.

Herausgegeben von der .TXT Publishing GmbH – einem Team aus Fachlektoren und Spezialisten, die aus eigener EDV- und Verlagspraxis die Informationsbedürfnisse von Computeranwendern kennen.

Systemanforderungen

Hardware
IBM-PC/XT/AT oder Kompatible mit mindestens 512 Kbyte freiem Arbeitsspeicher

Software
Power Basic 2.10

ISBN 3-87791-192-7

9 783877 911921 04900

DM 49,– sFr 46,– öS 382,–

Markt&Technik